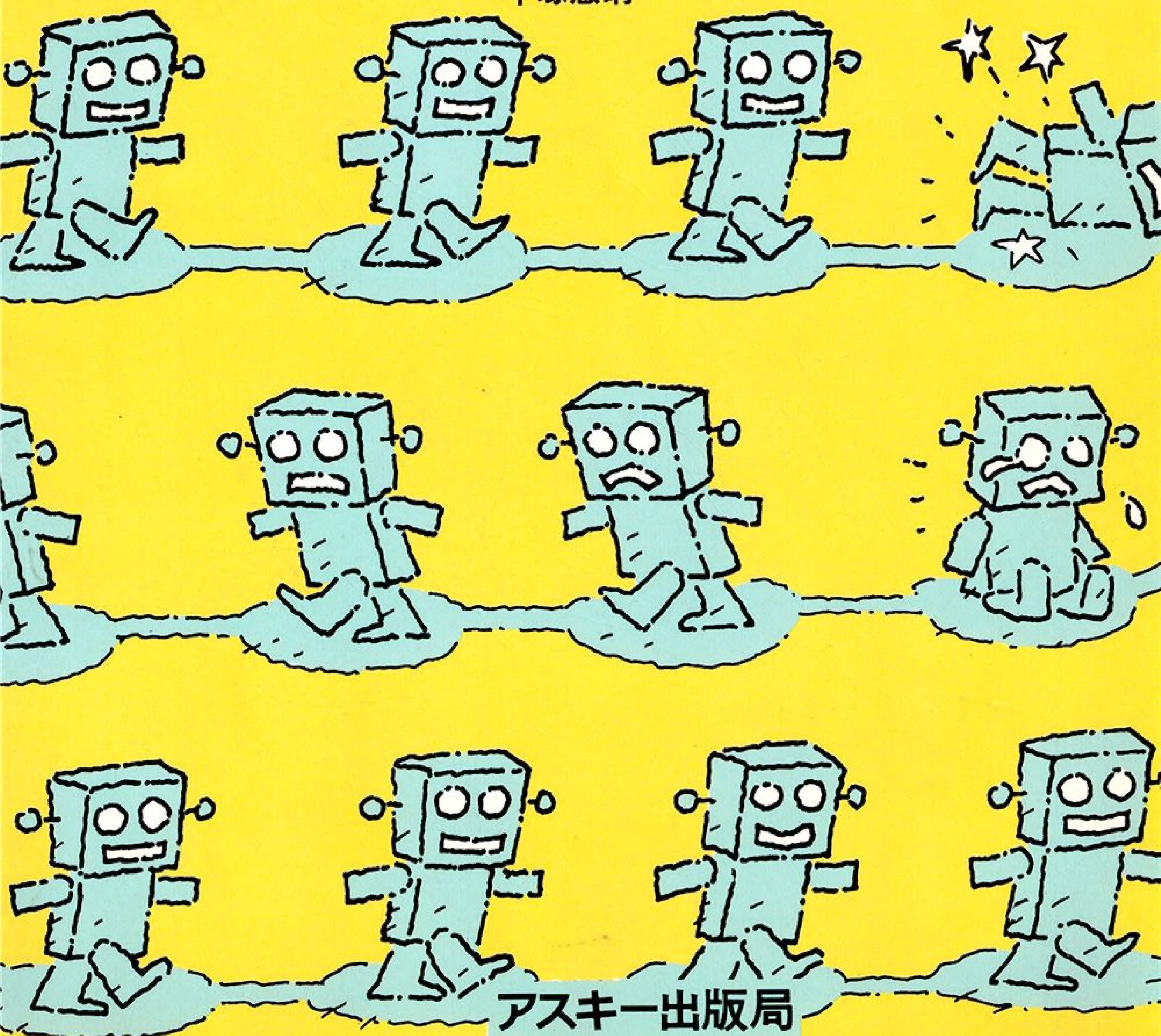


くじけちゃいけない!

マシン語入門

平塚憲晴



アスキー出版局

		く	じ	け	ち	ゃ			
		い	け	な	い				
		マ	シ	ン	語	入	門		

平 塚 憲 晴 著
MSXマガジン 監修

アスキー出版局

目次

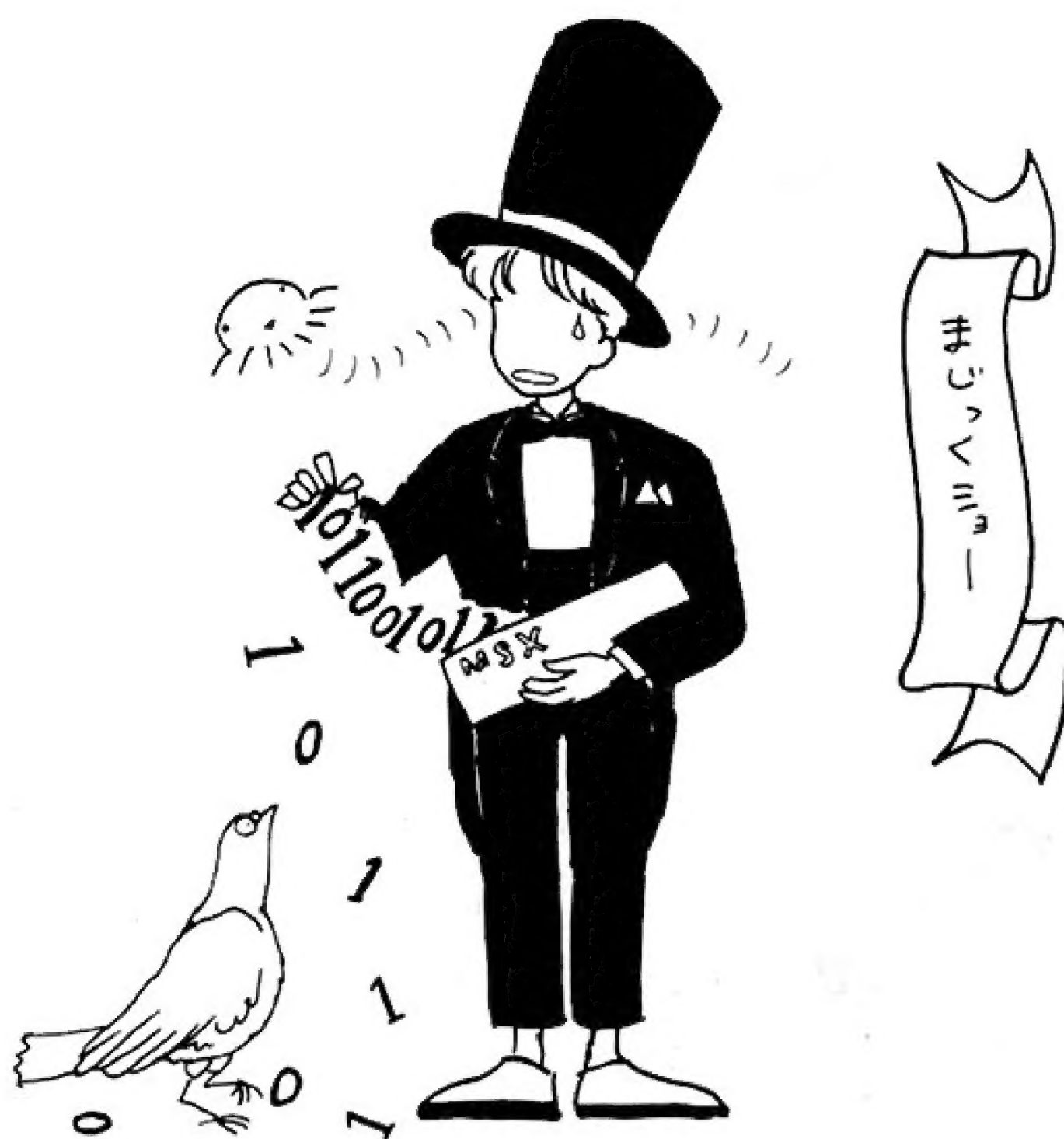
第1章 マシン語の正体を知ろう	5
1.1 マシン語なんてへっちゃら気分	6
1.2 2進数と、ついでの16進数	7
1.3 ダンプリストの正体、1バイトと1ビット	8
1.4 データを入れる引き出し、メモリとアドレス	12
1.5 メモリの種類、ROMとRAM	14
1.6 メモリの大きさを表す	16
1.7 計算するときは、レジスタを使う	17
1.8 マシン語の謎をとときあかせ	19
1.9 だから今アセンブラ	22
1.10 プログラムを作る前に一言	23
こらむ BASICはマシン語で動作している	25
第2章 マシン語モニタとアセンブラ	27
2.1 マシン語モニタとアセンブラ	28
2.2 マシン語モニタと アセンブラを入力する前の注意	28
2.3 マシン語モニタの入力	29
2.4 マシン語モニタの使い方	30
2.5 マシン語モニタのコマンドの説明	31
2.6 アセンブラの入力	37
2.7 アセンブラの使い方	39
2.8 おっと！ アセンブルエラーが出てしまった	43
こらむ 入力ミスは許さない。 チェックサムって何だろう	46

第3章 マシン語の基本命令を覚えよう	47
3.1 代入してみよう	48
3.2 計算してみよう	67
第4章 プログラムの流れを変えてみよう	83
4.1 分岐してみよう	84
4.2 判定してみよう	94
4.3 ループしてみよう	109
こらむ 暴走はどうしておこるのか	115
第5章 入出力装置をコントロールしてみよう	117
5.1 MSXの手足、入出力装置	118
5.2 画面に文字を出してみよう	121
5.3 キーボードから文字を入力してみよう	128
5.4 カーソルを移動させてみよう	129
5.5 CtrlとSTOPキーが 押されているか調べよう	130
5.6 ジョイスティックとトリガボタンの 状態を調べよう	132
5.7 CAPランプをつけたり、 消したりしてみよう	134
5.8 VPOKE、VPEEKしてみよう	136
付 録	139
●マシン語命令表	140
●マシン語モニタプログラム	146
●アセンブラ・ダンプリスト	151
索 引	157

表紙イラスト	ケロヨン村田
本文イラスト	石田育絵
協力	高橋秀樹

マシン語の 正体を 知ろう

第
1
章



1.1 マシン語なんてへっちゃら気分

みなさんはマシン語というと、まっ先にマシン語ダンプリストやBASIC プログラム中のマシン語データを思い浮かべるに違いありません。しかし、マシン語っていうのはいったい何なのでしょう。あのアルファベットまじりの数字が、整然と並んでいるようすを見ると、数字ぎらいの人でなくとも、なんだか身体がかゆくなってくるというものです。しかも、これが意味のある言葉（言語）だっていうんですから、英語もろくにわからないのにこんなのわかるはずがないと思いこんでしまうのも当然です。

先に結論を言ってしまうと、マシン語というのはちょっと複雑なパズルのようなものです。ちえの輪をはずすような気持ちで取り組みれば、ふとしたことですべてが見えてくるもの。もちろん英語よりもズツと簡単であると断言しましょう。変にできないできないと思わずに、気楽にやることが大切です。

マシン語に取りかかる前に、いくつか知っておいて欲しいことがあります。それは、「英語はアルファベットでできている」とか、「アルファベットにはA～Zがある」といったたぐいの、ごく初歩的なことですから、パラパラと見て知っていることであれば読み飛ばすなりしてください。

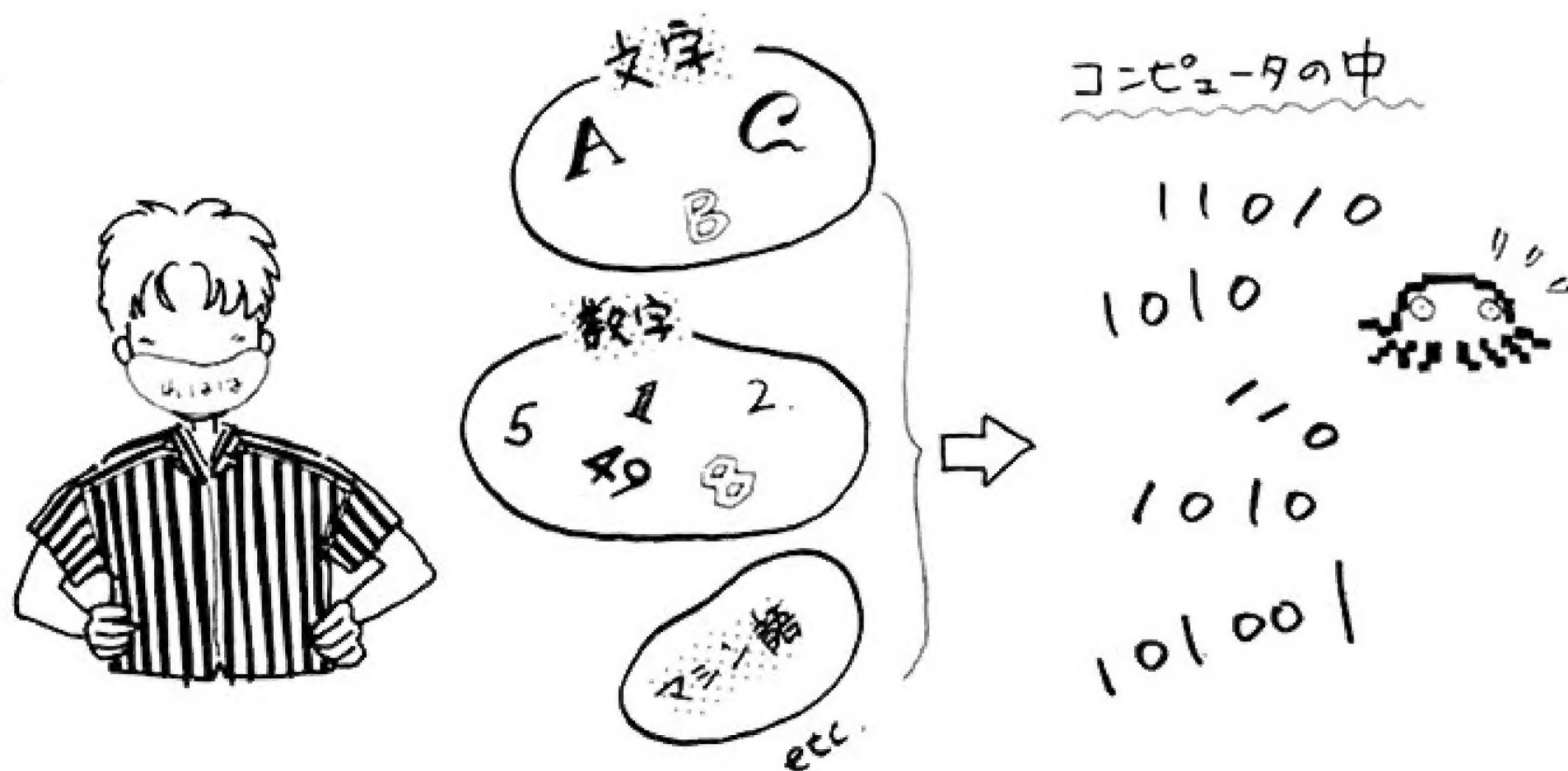
この章に書かれていることがすべてわかってしまったら、すでにマシン語の世界の30%ぐらいを理解できたようなものですから、このような人は、どんどんマシン語のプログラムを作ってみるべきです。すぐに2章へいきましょう。

その他の人でも「マシン語ダンプリストというのはこんなもので、ついでにマシン語というのは、よくわからないけどこんなものか」と思ったら、2章へいってしまいましょう。

なぜならマシン語を覚えるには、やっぱりじかに触れてみるのが一番ですからね。

1.2 2進数と、ついでの16進数

マシン語を理解するには、一般的に2進数から入るのが素直な方法です。それは、マシン語のプログラムや、マシン語であつかうデータが、コンピュータの中ではすべて2進数の形であつかわれているからです。



2進数は“0”と“1”の2つの文字だけで数を表す方法です。忘れてしまった人もいるかもしれませんので、普段使っている10進数と比べてみます（表1-1）。


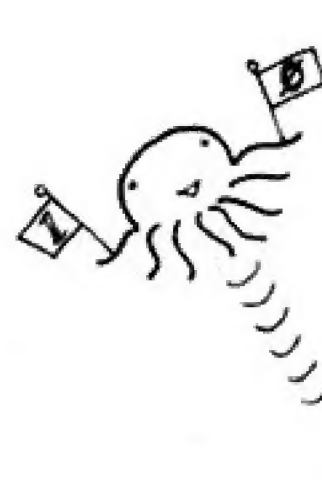
▶ 10進数	▶ 2進数
 <div data-bbox="679 2037 1004 2654"> 0 1 2 3 4 5 6 7 8 9 10 ← ケタあがり 11 12 ⋮ </div>	<div data-bbox="1172 2037 1657 2654"> 0 1 10 ← ケタあがり 100 101 110 111 1000 1001 1010 1011 1100 1101 1110 1111 10000 ⋮ </div> 

表1-1 10進数と2進数の対応表

2進数自体はどうということもないので、このぐらいにしておきます(まったく2進数がわからないときは、簡単ですから算数の本などで勉強しましょう)。

BASIC を使って10進数を2進数の形にするには次のようにします。


- 10進数を2進数の形にして表示

PRINT BIN\$ (<10進数>)

例) PRINT BIN\$ (12) 

- 2進数を10進数の形にして表示

PRINT <2進数>

例) PRINT &B1100 

1.3 ダンプリストの正体、1バイトと1ビット

マシン語は2進数で記憶されていると言いましたが、ただの2進数ではなく、正確に言うと「8ケタの2進数」で記憶されています。ですから、2進数で“11101”という数を表す場合も、マシン語では“00011101”としなければいけません。

それでは、なぜ8ケタの2進数にする必要があるのでしょうか。コンピュータの中では、数値をスイッチのオン/オフで記憶しています。スイッチといっても、家の電気をつけたりするような機械的なものではなく、トランジスタやコンデンサなどを使った、とっても小さい電氣的なスイッチです。そして、スイッチが入っているところ(電気が通っているところ)が1で、そうでないところが0を表します。2進数で表現するのはこのためで、8ケタにする理由は、要するにそのスイッチを8個使って1つの数値を記憶しているからです(図1-1)。

1.3 ダンプリストの正体、1バイトと1ビット

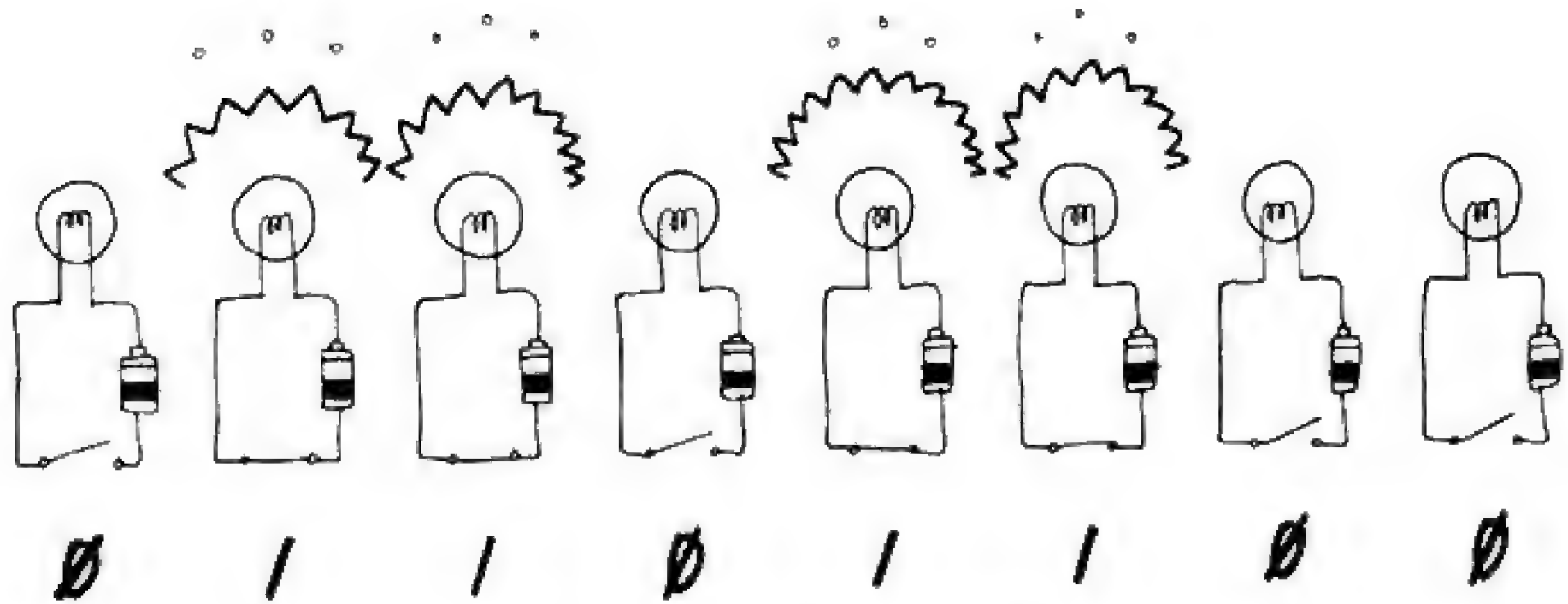


図1-1 数値が記憶される様子

そして、このとき2進数のそれぞれのケタを「1ビット(bit)」、さらに8ケタ(8ビット)をひとまとめて「1バイト(byte)」といいます(図1-2)。

1バイトで表すことができる数は“00000000”から“11111111”までですから、これを10進数の形にすると0~255になります。マシン語では、この0~255までの数値が基本となります。

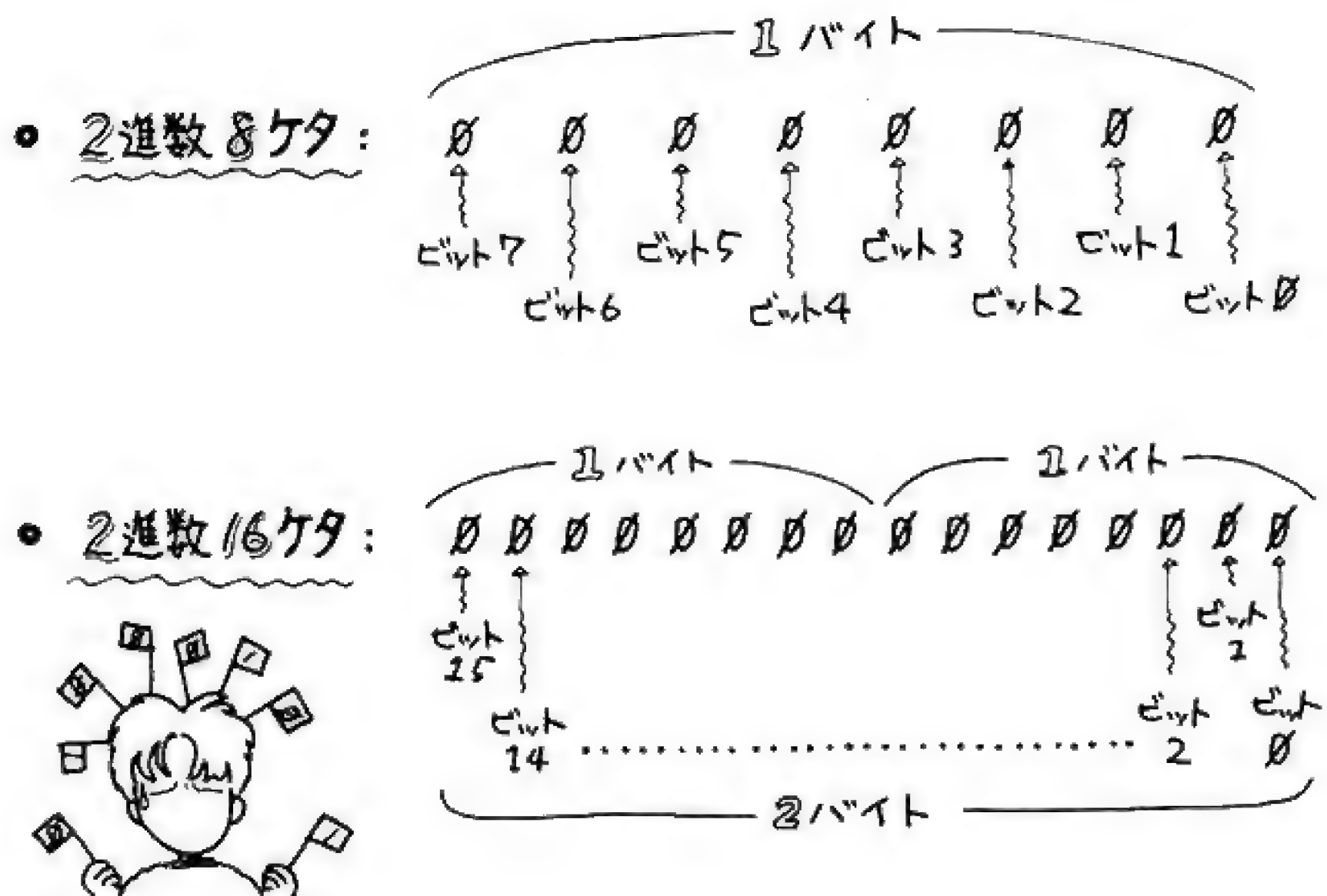


図1-2 1バイトと1ビットの関係

第1章 マシン語の正体を知ろう

ところでみなさんは、友達に「1 バイトのデータ教えてよ」と言われたとき、「00011101 だよ」なんて答えるのは大変だと思いませんか。それに、教えてもらった方だって、何だかよく聞き取れないんじゃないでしょうか。そこで、この1 バイトを上位4ビットと下位4ビットの2つに分けてみます。

0 0 0 1	1 1 0 1
上位4ビット	下位4ビット

2進数4ケタで表せる数値は16通りあります。そこで2進数4ケタで表せる数値を表1-2のように16進数と対応させてみます。

● 10進数	● 2進数4ビット	● 16進数
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F


 私たちが普段使っている10進数では、1つのケタに0~9までの10種類の数字を使っています。ところが16進数では1つのケタに16種類の数字を用意する必要があるとわかります。そこで一般的にアルファベットのA~Fを使って足りない数字をまぎらうんだヨ。

表1-2 4ケタの2進数と16進数の対応表

どうですか？ 8ケタの2進数を2ケタの16進数で表すことができますね。これが、アルファベットまじりの数字の正体です。

0 0 0 1	1 1 0 1
↓	↓
1	D

一般的に2進数8ケタ(1バイト)は16進数2ケタの数値で表されますが、これは何も16進数でなくても全然かまいません(実際に8進数などを使っている場合もある)。ただ、2進数のビットのイメージがつかみやすく、数値としてもあつかいやすいので多用されているわけです。

英語で10進数のことを「デシマル(Decimal)」、2進数のことを「バイナリ(Binary)」、16進数のことを「ヘキサデシマル(Hexadecimal)」といいます。ですからこの場合も「ヘキサの1Dだよ」なんて人に言うと、ちよつとカッコいいですね。

16進数の数字を書く場合“F9H”のように数字の後に“H”を、2進数の数字を書く場合“11111010B”のように“B”を付けたりしますが、これは“Hexadecimal”または“Binary”のイニシャルをとったものです(BASICでの“&H”や“&B”も同じです)。

なお、2進数や16進数の数字を読むときは、“111110101(イチイチイチイチゼロイチゼロイチ)”、“F9(エフキュー)”のように、1ケタごとに区切って言います。

BASICを使って10進数を16進数、2進数を16進数の形にするには次のようにします。

- 10進数を16進数の形にして表示

PRINT HEX\$ (<10進数>)

例) PRINT HEX\$ (12) 

- 16進数を10進数の形にして表示

PRINT <16進数>

例) PRINT &HFF 

- 16進数を2進数の形にして表示

PRINT BIN\$ (<16進数>)

例) PRINT BIN\$ (&HFF) 

- 2進数を16進数の形にして表示

PRINT HEX\$ (<2進数>)

例) PRINT HEX\$ (&B11111111) 

1. 4 データを入れる引き出し、メモリとアドレス

メモリがデータを記憶しておくところであることは知っていると思いますが、どのようなデータをどのくらい覚えられるのかわかりますか？

なんと、1バイトのデータを最高65536個も記憶することができます。1文字は1バイトで表すことができるので、これは400字づめの原稿用紙約160枚分にあたります。

メモリはこれだけたくさんありますが、その中の1バイトを指定する方法がなければ、あるひとつのメモリに対して、データを読んだり書いたりできません。そこで、メモリには0000H~FFFFH (0~65535)までの番号が順番にふられていて、どのメモリでもその番号で直接指定できるようになっています(図1-3)。この番号のことを「アドレス(または番地)」といいます。アドレスは0000H番地からFFFFH番地まで常に存在し、そこには1バイトの数値が必ず現れています。

アドレスというと、BASICの行番号と同じようなものだと思っている人がいるようですが、アドレスとはその名のとおり「住所」。つまり、メモリとしての回路が存在する実際の場所を示すものですから、行番号のようにコロコロとかえるわけにはいきません。たとえば、みなさんが「××市5番地」という場所に住んでいたとき、いくら「ここを××市2番地という住所にしたい」と言っても、そんなことはできないのと同じことです。

なお、ここではアドレスが4ケタの16進数、つまり2バイトで表すことができることも覚えておいてください。

1.4 データを入れる引き出し、メモリとアドレス

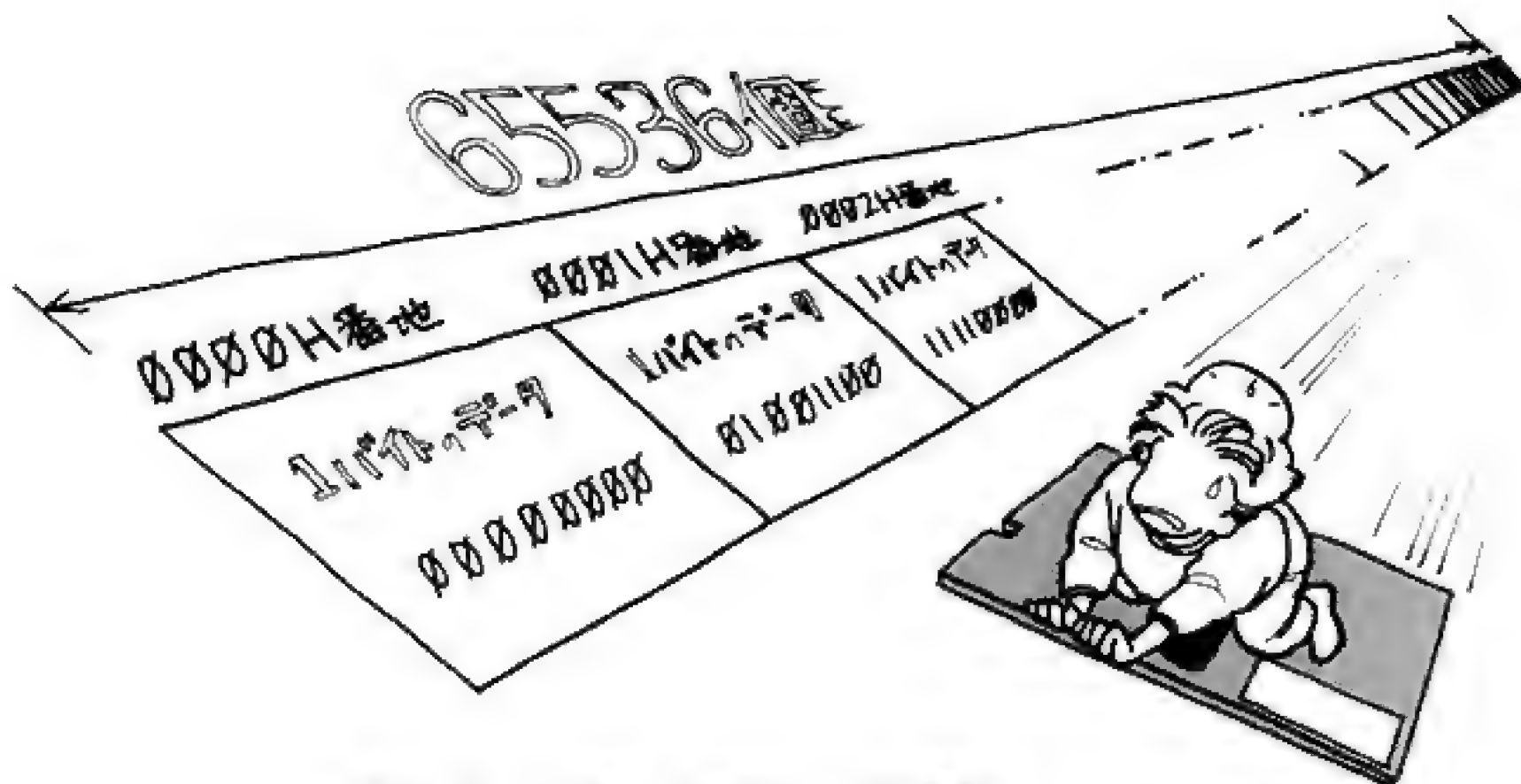


図1-3 メモリとアドレスの関係

ここで、もう1度マシン語ダンプリストを思い出してください。1番左にあるのがアドレスで、右に8つ並んでいるのがメモリに記憶されているデータです。ちょっと見ると、1つのアドレスに8つのデータが記憶されているような気もしますが、そうではなく、単に右の7つのデータのアドレスが省略されているだけなのです。1度にたくさんのデータを見ることができて便利ですね（図1-4）。

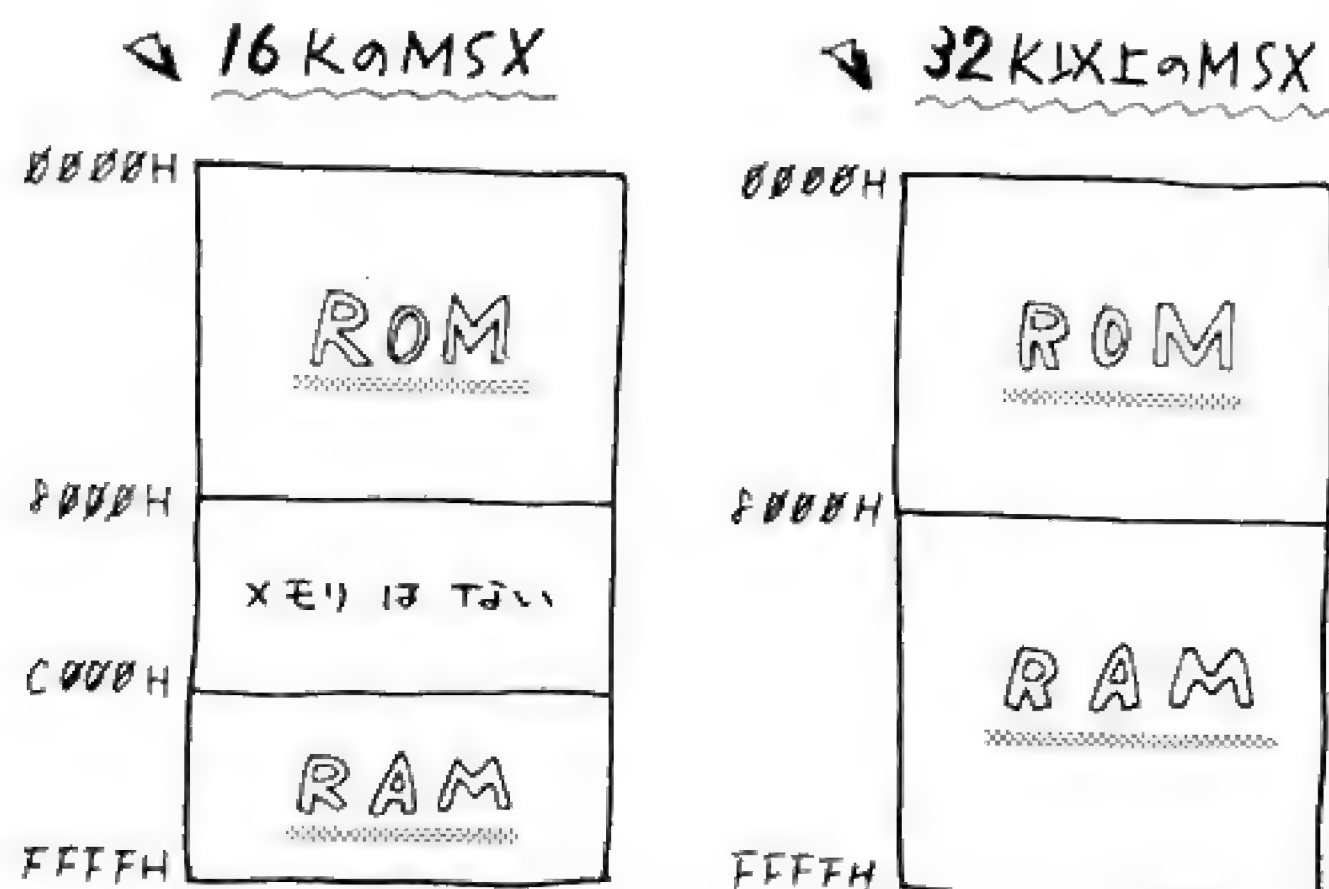
		D001番地の Xメモリの内容							
		D000番地の Xメモリの内容		D002番地の Xメモリの内容		D007番地の Xメモリの内容			
D000		01	04	A1	C5	80	11	63	81
D008		23	00	00	00	63	75	51	F3
D010		49	36	25	11	00	10	00	A1
D018		C3	D4	22	12	3A	3A	63	21
		D01F番地の Xメモリの内容		D01A番地の Xメモリの内容		D01F番地の Xメモリの内容			
		D019番地の Xメモリの内容							

図1-4 ダンプリストの意味

1.5 メモリの種類、ROMとRAM

メモリには「ROM(ロムまたはリードオンリーメモリ)」、と呼ばれているものと「RAM(ラムまたはランダムアクセスメモリ)」と呼ばれているものの2種類があります。ROMは「データの書きかえができないが、電源を切ってもデータが消えない」というガンコ者のメモリで、RAMは「データの書きかえはできるが、電源を切ってしまうとデータが消えてしまう」というナンジャク者のメモリです。

MSXでは、どのアドレスのメモリがROMまたはRAMか、図1-5に示します。



メモリがない所には、データを書くことはできません。
 手元、意味のあるデータがあらかじめ入っているわけでも
 ありません。要するに何にも使えないワケです。

図1-5 16Kと32K RAMのMSXのメモリ配置

VRAM(ビデオRAM)というメモリがありますが、これはプログラムやデータを記憶するRAMとはまったく別にあるメモリです。VRAMは、おもに画面に文字やグラフィックを表示するために存在します。つまり、VRAMというメモリにデータを書き込むことにより、実際にそのデータが画面に何らかの形で出てきたりするので

VRAMは5章でもう少し詳しく説明します。

なお、VRAMと普通のRAMを区別するために、RAMを「メインRAM」とすることがあります。

メモリが、実際どのようなことに使われているかは、マニュアルなどに載っている「メモリマップ」という図を見るとわかります。

▷ Xメモリマップ (32K以上のMSXの場合) ◁

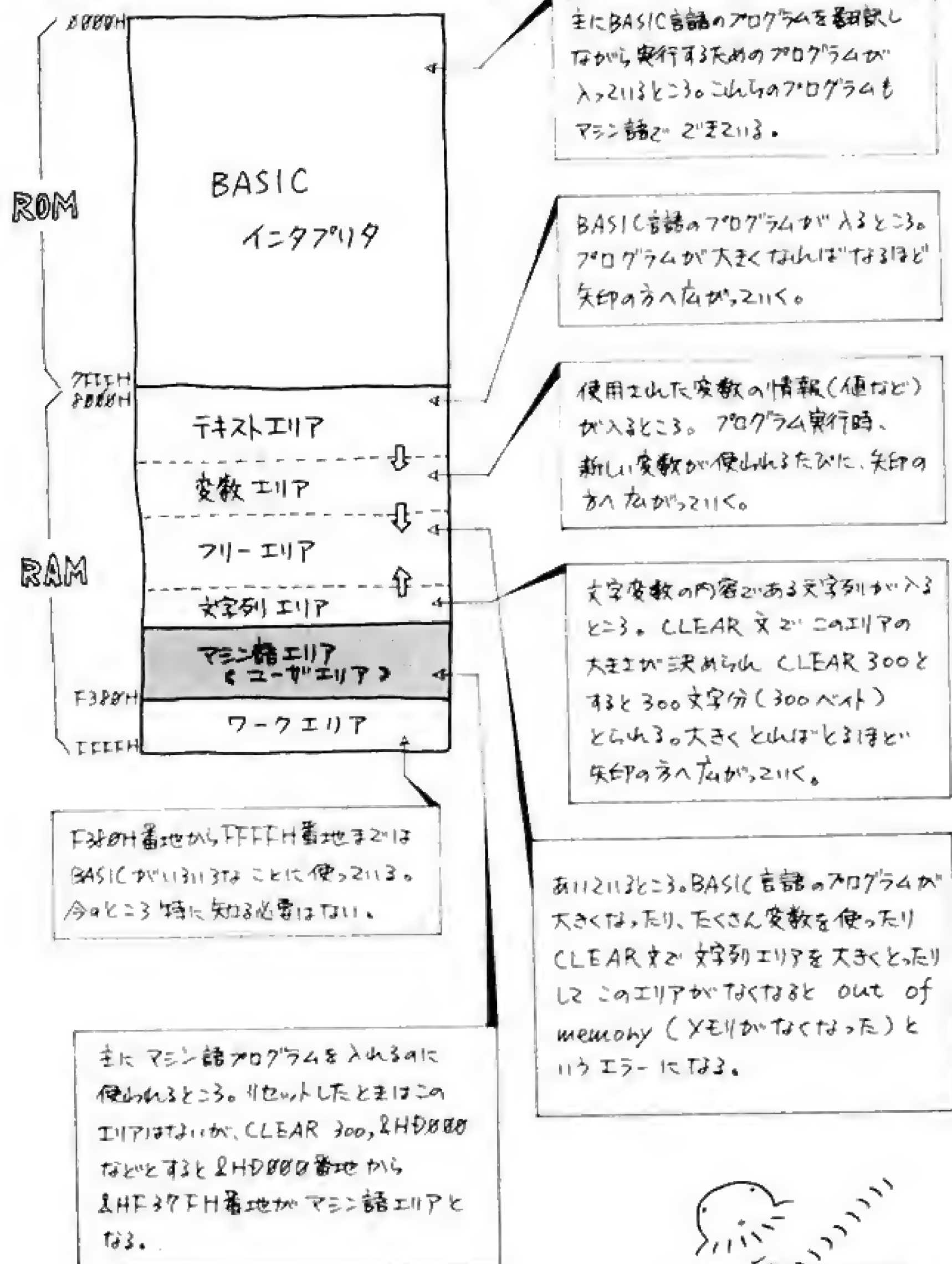


図1-6 32KMSXのメモリマップと、その使われ方

BASIC を使っているときは、だいたい図 1-6 のように使われています。

メモリマップで見るように、メモリは実にさまざまなことに使われているわけですが、実際はマシン語であろうが、他のデータであろうが、どれもただの 1 バイトの数値にすぎません。つまり、そのデータをどう使うかでそのデータの持つ意味も変わってくるのです。

1.6 メモリの大きさを表す

メモリの大きさ(容量)を表すとき、16K バイト、32K バイトなどといいますが、1K (ケー) バイトは 1024 バイト (400H バイト) に相当します。

重さの単位が 10 進数 3 ケタごとに、g、kg、t、とあるのと同じように、メモリの大きさも 2 進数で表した場合、10 ケタごとに、バイト、K (ケー) バイト、M (メガ) バイト、G (ギガ) バイトというように単位が変わっていきます(表 1-3)。メガ ROM というカートリッジがありますが、これは M (メガ) ビットなので、バイトに直すと (8 で割ると) 128K バイトということになります。

1K	1024	(400H) バイト
8K	8192	(2000H) バイト
16K	16535	(4000H) バイト
32K	32313	(8000H) バイト
64K	65536	(10000H) バイト

2進数で表したメモリの大きさ

単位	1	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192	16384	32768	65536	131072	262144	524288	1048576	2097152	4194304	8388608	16777216	33554432	67108864	134217728	268435456	536870912	1073741824	2147483648	4294967296	8589934592	17179869184	34359738368	68719476736	137438953472	274877906944	549755813888	1099511627776	2199023255552	4398046511104	8796093022208	17592186044416	35184372088832	70368744177664	140737488355328	281474976710656	562949953421312	1125899906842624	2251799813685248	4503599627370496	9007199254740992	18014398509481984	36028797018963968	72057594037927936	144115188075855872	288230376151711744	576460752303423488	1152921504606846976	2305843009213693952	4611686018427387904	9223372036854775808	18446744073709551616	36893488147419103232	73786976294838206464	147573952589676412928	295147905179352825856	590295810358705651712	1180591620717411303424	2361183241434822606848	4722366482869645213696	9444732965739290427392	18889465931478580854784	37778931862957161709568	75557863725914323419136	151115727451828646838272	302231454903657293676544	604462909807314587353088	1208925819614629174706176	2417851639229258349412352	4835703278458516698824704	9671406556917033397649408	19342813113834066795298816	38685626227668133590597632	77371252455336267181195264	154742504910672534362390528	309485009821345068724781056	618970019642690137449562112	1237940039285380274899124224	2475880078570760549798248448	4951760157141521099596496896	9903520314283042199192993792	19807040628566084398385987584	39614081257132168796771975168	79228162514264337593543950336	158456325028528675187087900672	316912650057057350374175801344	633825300114114700748351602688	1267650600228229401496703205376	2535301200456458802993406410752	5070602400912917605986812821504	10141204801825835211973625643008	20282409603651670423947251286016	40564819207303340847894502572032	81129638414606681695789005144064	162259276829213363391578010288128	324518553658426726783156020576256	649037107316853453566312041152512	1298074214633706907132624082305024	2596148429267413814265248164610048	5192296858534827628530496329220096	10384593717069655257060992658440192	20769187434139310514121985316880384	41538374868278621028243970633760768	83076749736557242056487941267521536	166153499473114484112975882535043072	332306998946228968225951765070086144	664613997892457936451903530140172288	1329227995784915872903807060280344576	2658455991569831745807614120560689152	5316911983139663491615228241121378304	10633823966279326983230456482242756608	21267647932558653966460912964485513216	42535295865117307932921825928971026432	85070591730234615865843651857942052864	170141183460469231731687303715884105728	340282366920938463463374607431768211456	680564733841876926926749214863536422912	1361129467683753853853498429727072845824	272225893536750770770699685945414569152	544451787073501541541399371890829138304	1088903574147003083082798743781658276608	2177807148294006166165597487563316553216	4355614296588012332331194975126633106432	8711228593176024664662389950253266212864	17422457186352049329324779900506532425728	34844914372704098658649559801013064851456	69689828745408197317299119602026129702912	139379657490816394634598239204052259405824	278759314981632789269196478408104518811648	557518629963265578538392956816209037623296	1115037259926531157076785913632418075246592	2230074519853062314153571827264836150493184	4460149039706124628307143654529672300986368	8920298079412249256614287309059344601972736	17840596158824498513228574618118689203945472	35681192317648997026457149236237378407890848	71362384635297994052914298472474756815781696	142724769270595988105828596944949513631563392	285449538541191976211657193889899027263126784	570899077082383952423314387779798054526253568	1141798154164767904846628775559596109052507136	2283596308329535809693257551119192218105014272	4567192616659071619386515102238384436210028544	9134385233318143238773030204476768872420057088	18268770466636286477546060408953537744840114176	36537540933272572955092120817907075489680228352	73075081866545145910184241635814150979360456704	146150163733090291820368483271628301958720913408	292300327466180583640736966543256603917441826816	584600654932361167281473933086513207834883653632	1169201309864722334562947866173026415669767307264	2338402619729444669125895732346052831339534614528	4676805239458889338251791464692105662679069229056	9353610478917778676503582929384211325358138458112	18707220957835557353007165858768422650716276916224	37414441915671114706014331717536845301432553832448	74828883831342229412028663435073690602865107664896	149657767662684458824057326870147381205730215329792	299315535325368917648114653740294762411460430659584	598631070650737835296229307480589524822920861319168	1197262141301475670592458614961179049645841722638336	2394524282602951341184917229922358099291683445276672	4789048565205902682369834459844716198583366890553344	9578097130411805364739668919689432397166733781106688	19156194260823610729479337839378864794333467562213376	38312388521647221458958675678757729588666935124426752	76624777043294442917917351357515459177333870248853504	153249554086588885835834702715030918354667740497707008	306499108173177771671669405430061836709335480995414016	612998216346355543343338810860123673418670961990828032	1225996432692711086686677621720247346837341923981665664	2451992865385422173373355243440494693674683847963331328	4903985730770844346746710486880989387349367695926662656	9807971461541688693493420973761978774698735391853325312	19615942923083377386986841947523957549397470783706650624	39231885846166754773973683895047915098794941567413301248	78463771692333509547947367790095830197589883134826602496	156927543384667019095894735580191660395179766269653204992	313855086769334038191789471160383320790359532539306409984	627710173538668076383578942320766641580719065078612819968	1255420347077336152767157884641533283161438130157225639936	2510840694154672305534315769283066566322876260314451279872	5021681388309344611068631538566133132645752520628902559744	10043362776618689222137263077132266265291505041257805119488	20086725553237378444274526154264532530583010082515610238976	40173451106474756888549052308529065061166020165031220477952	80346902212949513777098104617058130122332040330062440955904	160693804425899027554196209234116260244664080660124881911808	321387608851798055108392418468232520489328161320249763823616	642775217703596110216784836936465040978656322640499527647232	1285550435407192220433569673872930081957312645280999055294464	2571100870814384440867139347745860163914625290561998110588928	5142201741628768881734278695491720327829250581123996221177856	10284403483257537763468557390983440655658501162247992442355712	20568806966515075526937114781966881311317002324495984884711424	41137613933030151053874229563933762622634004648991969769422848	82275227866060302107748459127867525245268009297983939538845696	164550455732120604215496918255735050490536018595967879077691392	329100911464241208430993836511470100981072037191935758155382784	658201822928482416861987673022940201962144074383871516310765568	1316403645856964833723975346045880403924288148767743032621531136	2632807291713929667447950692091760807848576297535486065243062272	5265614583427859334895901384183521615697152595070972130486124544	10531229166855718669791802768367043231394305190141944260972249088	21062458333711437339583605536734086462788610380283888521944498176	42124916667422874679167211073468172925577220760567777043888996352	84249833334845749358334422146936345851154441521135554087777992704	168499666669691498716668844293872691702308883042271110175555985408	336999333339382997433337688587745383404617766084542220351111970816	673998666678765994866675377175490766809235532169084440702223941632	1347997333357531989733350754350981533618471064338168881404447883264	2695994666715063979466701508701963067236942128676337762808895766528	5391989333430127958933403017403926134473884257352675525617791533056	10783978666860255917866806034807852268947768514705351051235583066112	21567957333720511835733612069615704537895537029410702102471166132224	43135914667441023671467224139231409075791074058821404204942332264448	86271829334882047342934448278462818151582148117642808409884664528896	172543658669764094685868896556925636303164296235285616819769329057792	345087317339528189371737793113851272606328592470571233639538658115584	690174634679056378743475586227702545212657184941142467279077316231168	1380349269358112757486951172455405090425314369882284934558154632462336	2760698538716225514973902344910810180850628739764569869116309264924672	5521397077432451029947804689821620361701257479529139738232618529849344	11042794154864902059895609379643240723402514959058279476465237059698688	22085588309729804119791218759286481446805029918116558952930474119397376	44171176619459608239582437518572962893610059836233117905860948238794752	88342353238919216479164875037145925787220119672466235811721896477589504	176684706477838432958329750074291851574440239344932471623443792955179008	353369412955676865916659500148583703148880478689864943246887585910358016	706738825911353731833319000297167406297760957379729886493775171820716032	1413477651822707463666638000594334812595521914759459772987550343641432064	2826955303645414927333276001188669625191043829518919545975100687282864128	5653910607290829854666552002377339250382087659037839091950201374565728256	11307821214581659709333104004754678500764175318075678183900402749131456512	22615642429163319418666208009509357001528350636151356367800805498262913024	45231284858326638837332416019018714003056701272302712735601610996525826048	90462569716653277674664832038037428006113402544605425471203221993051652096	180925139433306555349329664076074856012226805089210850942406443986103304192	361850278866613110698659328152149712024453610178421701884812887972206608384	723700557733226221397318656304299424048907220356843403769625775944413216768	1447401115466452442794637312608598848097814440713686807539251551888826433536	2894802230932904885589274625217197696195628881427373615078503103777652867072	5789604461865809771178549250434395392391257762854747230157006207555305734144	11579208923731619542357098500868790784782515525709494460314012415110611468288	23158417847463239084714197001737581569565031051418988920628024830221222936576	46316835694926478169428394003475163139130062102837977841256049660442445873152	92633671389852956338856788006950326278260124205675955682512099320884891746304	185267342779705912677713576013900652556520248411351911365024198641769783492608	370534685559411825355427152027801305113040496822703822730048397283539566985216	741069371118823
----	---	---	---	---	----	----	----	-----	-----	-----	------	------	------	------	-------	-------	-------	--------	--------	--------	---------	---------	---------	---------	----------	----------	----------	-----------	-----------	-----------	------------	------------	------------	------------	-------------	-------------	-------------	--------------	--------------	--------------	---------------	---------------	---------------	---------------	----------------	----------------	----------------	-----------------	-----------------	-----------------	------------------	------------------	------------------	------------------	-------------------	-------------------	-------------------	--------------------	--------------------	--------------------	---------------------	---------------------	---------------------	---------------------	----------------------	----------------------	----------------------	-----------------------	-----------------------	-----------------------	------------------------	------------------------	------------------------	------------------------	-------------------------	-------------------------	-------------------------	--------------------------	--------------------------	--------------------------	---------------------------	---------------------------	---------------------------	---------------------------	----------------------------	----------------------------	----------------------------	-----------------------------	-----------------------------	-----------------------------	------------------------------	------------------------------	------------------------------	------------------------------	-------------------------------	-------------------------------	-------------------------------	--------------------------------	--------------------------------	--------------------------------	---------------------------------	---------------------------------	---------------------------------	----------------------------------	----------------------------------	----------------------------------	----------------------------------	-----------------------------------	-----------------------------------	-----------------------------------	------------------------------------	------------------------------------	------------------------------------	-------------------------------------	-------------------------------------	-------------------------------------	-------------------------------------	--------------------------------------	--------------------------------------	--------------------------------------	---------------------------------------	---------------------------------------	---------------------------------------	--	--	--	--	---	---	---	--	---	---	--	--	--	--	---	---	---	--	--	--	---	---	---	---	--	--	--	---	---	---	--	--	--	--	---	---	---	--	--	--	---	---	---	---	--	--	--	---	---	---	--	--	--	--	---	---	---	--	--	--	---	---	---	---	--	--	--	---	---	---	--	--	--	---	---	---	---	--	--	--	---	---	---	--	--	--	--	---	---	---	--	--	--	---	---	---	---	--	--	--	---	---	---	--	--	--	--	---	---	---	--	--	--	---	---	---	---	--	--	--	---	---	---	--	--	--	--	---	---	---	--	--	--	---	---	---	---	--	--	-----------------

1.7 計算するときは、レジスタを使う

BASICでは、「変数」を使ってデータの記憶や計算を行います。それに対し、マシン語では「メモリ」と「レジスタ」がかわりをつとめます。レジスタというならば「計算できるメモリ」で、メモリが「記憶するための脳」ならば、レジスタは「考えるための脳」にあたる非常に大切なものです。レジスタには図 1-7 のようなものがあり、それぞれ特徴があります。

メモリとレジスタの違いを、もう少しはつきりさせてみましょう。

まず、電卓を使って問題用紙に書いてある簡単な計算問題を解く、という状況を考えてみてください。普通は次のような手順で計算を行います。

問題 $1234+5678=$

- ① 初めに問題用紙の 1234 という数値を、電卓に入力する
- ② 電卓の $+$ ボタンを押して、次の数を足すことを知らせる
- ③ 問題用紙の 5678 という数値を入力する
- ④ 電卓の $=$ ボタンを押して、計算結果を出力させる
- ⑤ 結果を問題用紙に書き写す



第1章 マシン語の正体を知ろう

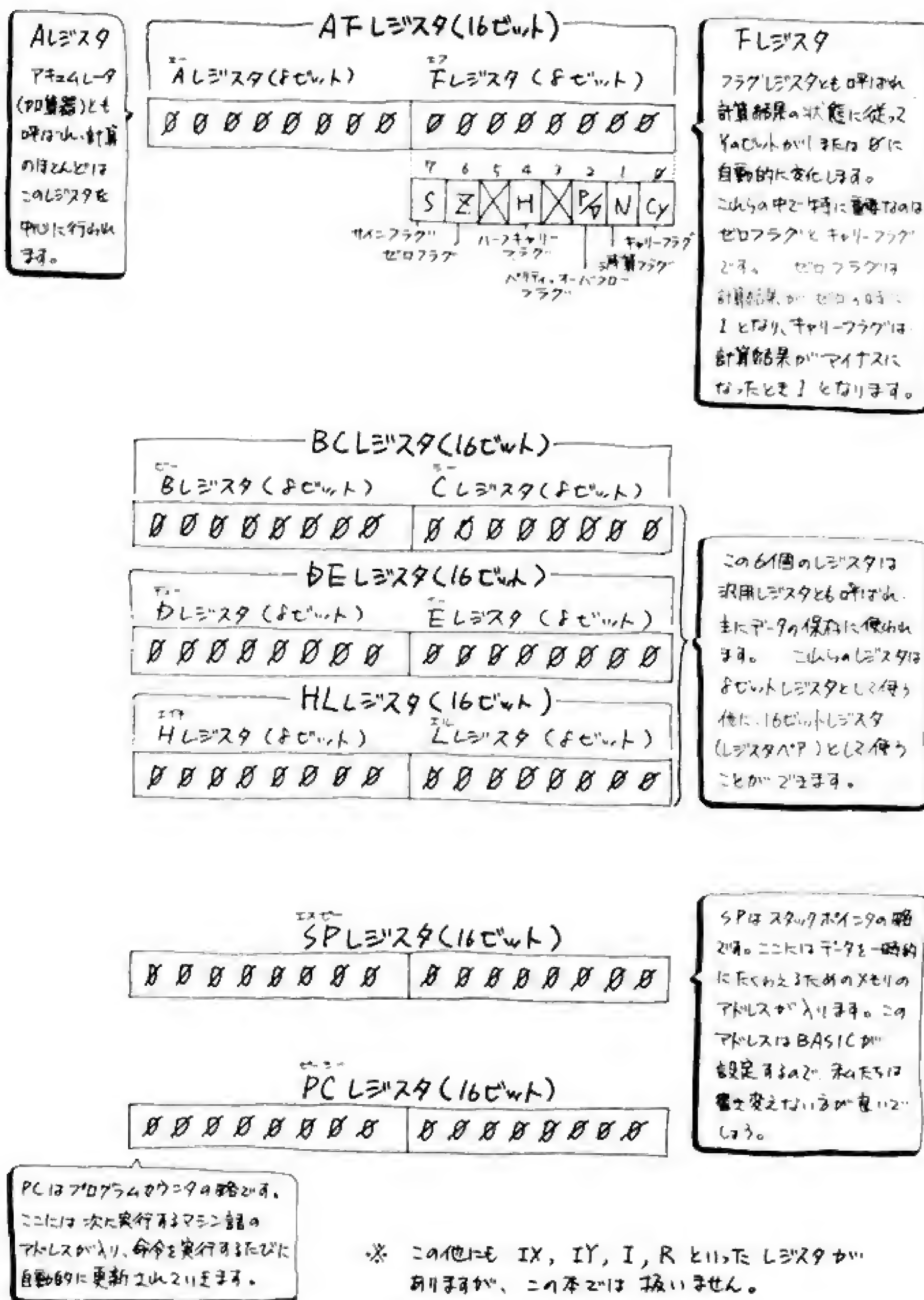


図1-7 レジスタ一覧

問題の書いてある用紙は、その問題および答えを覚えておくためだけにあり、電卓は計算をするためだけにあります。つまり、データを記憶するための問題用紙に相当するものがメモリで、計算するための電卓に相当するものがレジスタなのです。では実際のプログラムのさわりとして、以下の問題を解くプログラムの手順をあげてみます。

問題 D100H 番地の内容と数値の 2 を加えて、結果を D102 番地に書き込む。

- ① D100H 番地に記憶されている内容を、A レジスタに入力する
- ② 数値の 2 を、B レジスタに入力する
- ③ A レジスタと B レジスタの内容を足す（結果は A レジスタに入ることになっている）
- ④ A レジスタの内容を D102H 番地に書き込む

このように、数値に対し何かをする場合は、必ずレジスタを介して行う必要があります。

1.8 マシン語の謎をときあかせ

最後に、マシン語のプログラム自身について話しておきましょう。まず、マシン語のプログラムがメモリにあるようすを見てみます(図 1-8)。

この 2 進数(16 進数)の並びがマシン語の命令の集まり、つまりマシン語のプログラムで、これには実は図 1-9 のような意味がかかれており、このプログラムを実行すると図 1-10 のようなことが行われます。なお、メモリやレジスタは、変数のように初期値が必ず 00H だとは限りません。ここでは、たまたま D100H 番地が A1H で、A レジスタ、B レジスタ、D101H 番地、D102H 番地が 00H の場合で行ってみました。

アドレス	内容	
	2進数	16進数
CFFE H	= 00000000 (00H)	
CFFF H	= 00000000 (00H)	
D000 H	= 00111010 (3AH)	
D001 H	= 00000000 (00H)	
D002 H	= 11010001 (D1H)	
D003 H	= 00000110 (06H)	
D004 H	= 00000010 (02H)	
D005 H	= 10000000 (80H)	
D006 H	= 00110010 (32H)	
D007 H	= 00000010 (02H)	
D008 H	= 11010001 (D1H)	
D009 H	= 00000000 (00H)	
D00A H	= 00000000 (00H)	

マシン語として
意味のある
データの集まり。

図1-8 メモリにあるマシン語プログラムのような

3バイト 命令	D000H = (3AH)	次の2バイトで表されるアドレスの内容をレジスタAに 入れるという命令。
	D001H = (00H)	アドレスを表す。(D100H番地)
	D002H = (D1H)	
2バイト 命令	D003H = (06H)	次の1バイトの値をレジスタBに入れるという命令。
	D004H = (02H)	値を表す。
1バイト 命令	D005H = (80H)	レジスタAの内容にレジスタBの内容を足せという 命令。結果はレジスタAに入ることに決まっている。
3バイト 命令	D006H = (32H)	レジスタAの内容を次の2バイトで表されるアドレ スに入れるという命令。
	D007H = (02H)	
	D008H = (D1H)	

↑
なぜこのように分けることが
できるかというと、命令により
その後続くデータのバイト数が
決まっているから。

図1-9 マシン語プログラムの意味

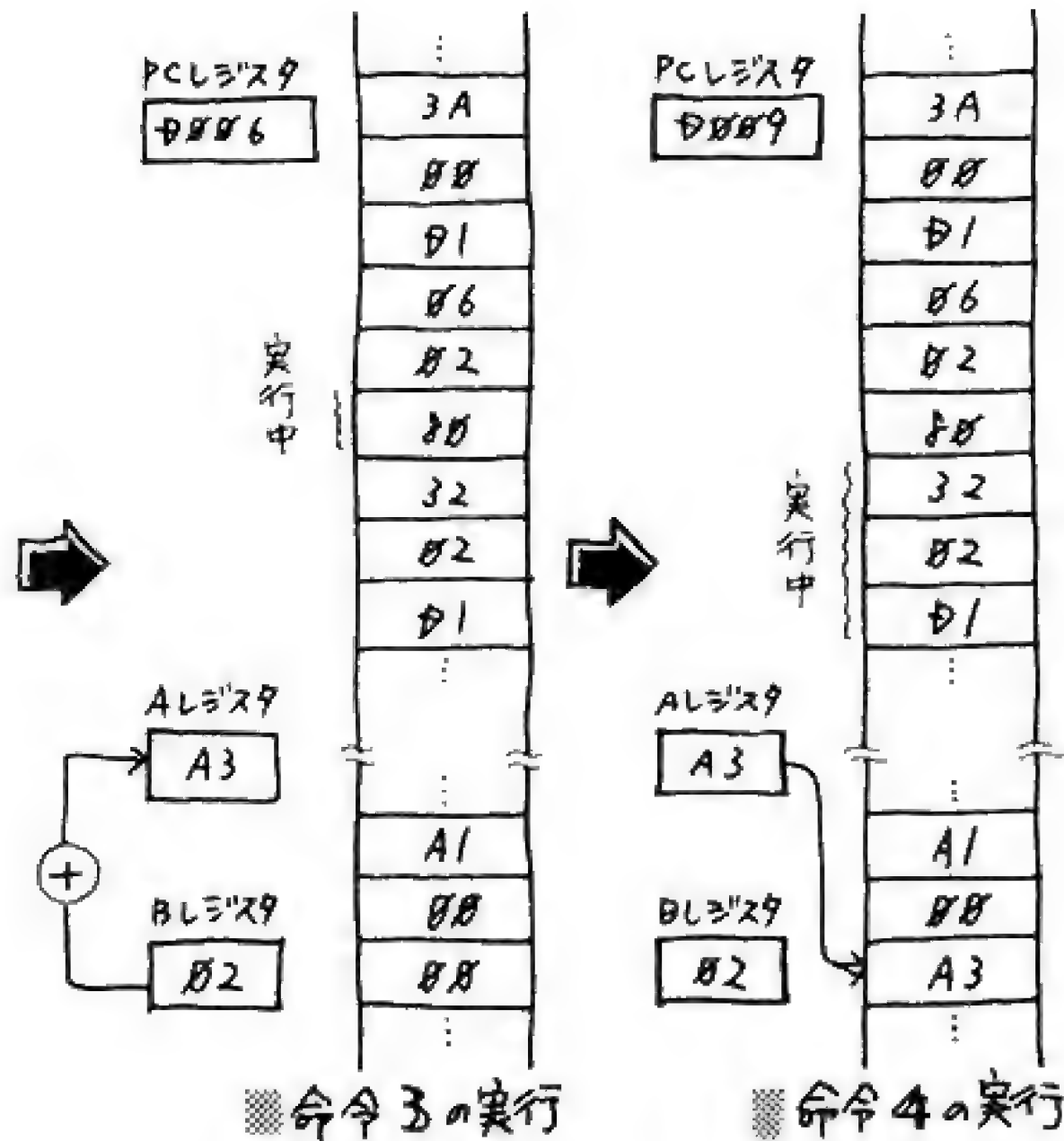
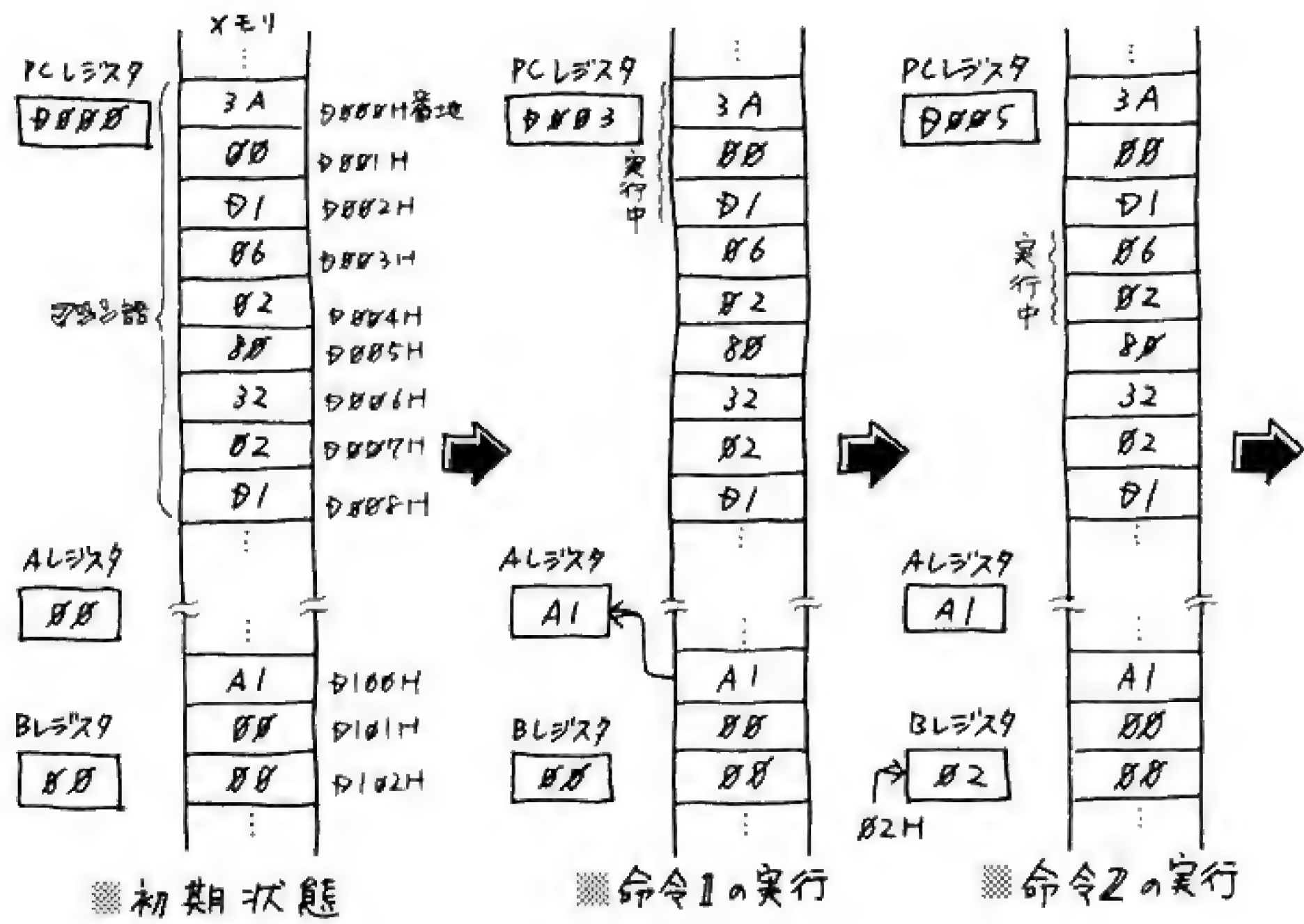


図1-10 マシン語プログラムの実行

1.9 だから今アセンブラ

だいたいマシン語がどんなものかはわかりましたが、実際にマシン語を数字だけで考えるのは、非常にたいへんです。そこで一般にマシン語のプログラムは「ニーモニック」という単語（のようなもの）を使った「アセンブリ言語」で考えます。では、ためにさきほどのマシン語プログラムを、ニーモニックを使って置きかえてみます（図 1-11）。

アドレス	マシン語	アセンブリ言語
D000	3A00D1	LD A, (0D100H)
D003	0602	LD B, 2
D005	80	ADD A, B
D006	3202D1	LD (0D102H), A

図 1-11 マシン語とアセンブリ言語の対応

ただの数字の集まりに比べれば、ずうっとわかりやすいですね。私たちがマシン語プログラムを作るときは今とは逆に、初めはアセンブリ言語でプログラムを作り、それをマシン語に変換していけばいいわけです。

アセンブリ言語のプログラムをマシン語に変換することを「アセンブル」といい、人間が自分で一生懸命アセンブルすることを「ハンドアセンブル」といいます。しかしハンドアセンブルは非常に大変な作業です。そこで、この作業を自動的に行うために「アセンブラ」というプログラムがあり、これを使えばアセンブリ言語で書いたプログラムをマシン語のプログラムに、自動的に変換することができます。

アセンブラを中心に見たとき、アセンブリ言語で書かれたプログラムを「アセンブラのソースプログラム」といい、アセンブルした結果生成されるマシン語のプログラムのことを「オブジェクトプログラム」といいます(図 1-12)。なお、アセンブラのソースプログラムを「アセンブラのプログラム」と省略する場合があります。ソースには「源」、オブジェクトには「目的」という意味があります。

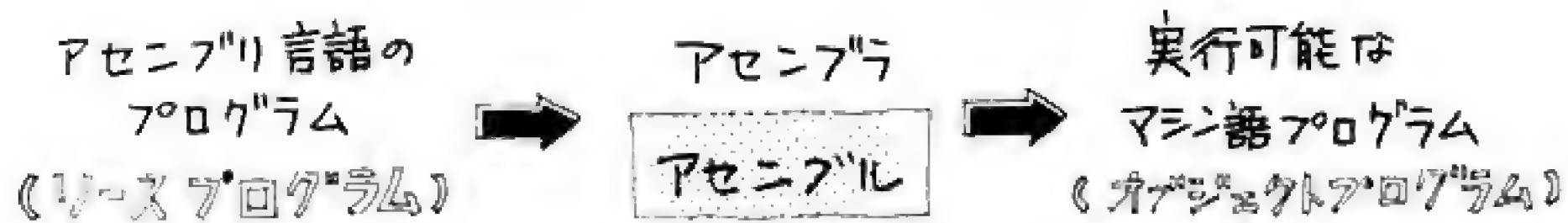


図1-12 アセンブラをつかったプログラムの開発

アセンブリ言語とマシン語は完全に 1 対 1 に対応しているので、アセンブリ言語を理解することは、そのままマシン語を理解することにつながります。ですから、本書ではすべてアセンブリ言語で話を進めていきます。

1.10 プログラムを作る前に一言

マシン語のプログラムを組む前に知っておいてもらいたいことは、これですべてです。何となくマシン語というものがわかりかけてきたでしょうか。2 章でアセンブラを入力したら、いよいよ 3 章から、マシン語の命令を紹介していきます。

本書でのマシン語の説明のしていき方ですが、ただ全命令を淡々と解説していくというような方法はとりません、なぜなら、単語だけ知っていても、英語が使いこなせないのと同様に、命令だけ覚えても、その組み合わせ方がわからなければ、プログラムは作れないからです。

さいわいにして、みなさんは BASIC 言語でちょっとしたプログラムなら作れると思います。それならば、マシン語のプログラムを直接考えるより、BASIC で簡単な「処理の流れ」を考えてから、それをマシン語に移植していく方が、まちがいも少なく、なれば BASIC の要領で、マシン語のプログラムが作れるようになるのではないのでしょうか。

ここでは、そのような見解を持って、たとえば判定文(IF～THEN 文)に相当するようなマシン語のプログラムを作るには、どの命令をどのように組み合わせればよいか、という方向で話を進めていき

たいと思います。

ただし、BASIC とマシン語はあくまでも違うものですから、PLAY 文をマシン語にするなどは、そうやすやすとできるものではありません。本書でとりあげるものは、以下にあげた BASIC の基本命令だけです。しかし、これらをマスターすれば、マシン語に対する恐怖心は、すでにどこかに飛んでしまっていることでしょう。ガンバってくださいね。

第3章	代入	A=3
	計算	A+3
第4章	分岐	GO TO, GOSUB~RETURN
	判定	IF~THEN
	ループ	FOR~TO~NEXT
第5章	文字表示	PRINT "A"
	文字入力	A\$=INPUT\$
	その他	STICK, VPOKE など

BASICはマシン語で動作している

コンピュータはマシン語のプログラムしか実行できません。ですから、なにげなくいつも使っている BASIC も、最終的にはマシン語で動作しています。ここでは、そのあたりにちよつとだけ触れてみましょう。

BASIC のプログラムは、ROM に記憶されている「BASIC インタプリタ」というマシン語プログラムが実行してくれています。では、実際どのように実行しているのか知るために、入力した BASIC のプログラムが、メモリに入っているようすを見てみましょう。次の BASIC プログラムを入力し、モニタの D コマンドで、32K 以上の RAM の MSX は 8000H 番地から、16K RAM の MSX は C000H 番地からメモリダンプしてみましょう(モニタについては2章をご覧ください)。

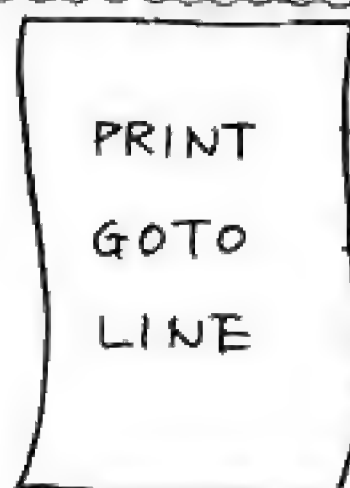
```
100 '= test =
110 '
120 A$="123 ABC abc"
130 PRINT A$
```

8000	00	11	80	64	00	3A	8F	E6	..#d.:っC100行
8008	3D	20	74	65	73	74	20	3D	= test =	
8010	00	19	80	6E	00	3A	8F	E6	..#n.:っC110行
8018	00	2E	80	78	00	41	24	EF	..#x.A\$120行
8020	22	31	32	33	20	41	42	43	"123 ABC	
8028	20	61	62	63	22	00	37	80	abc".7130行
8030	B2	00	91	20	41	24	00	00	..# A\$..	

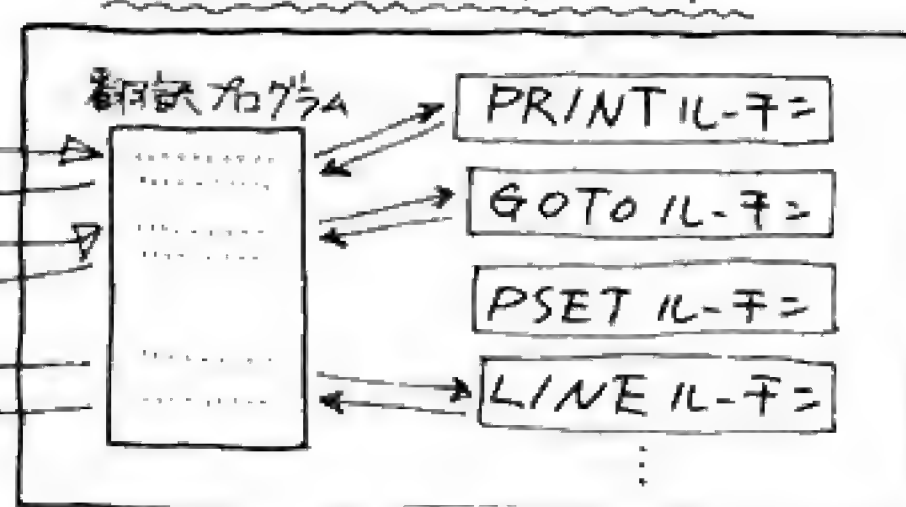
ちよつとグチャグチャしてますが、雰囲気的には BASIC のプログラムしてますね。BASIC のプログラムはこのように、LIST したときとは違う形でメモリに入っています。

ここで注意してもらいたいことは、これはマシン語ではなく、BASIC 言語の形をしたただのデータだということです。ですから、このデータのとおりには実行するためには、このデータを読み取り、それが何をやる文なのか判断し、それに応じたマシン語プログラムを実行させなければなりません(図)。インタプリタは実際にこのようなことをやっており、私たちはこのおかげで、BASIC という高級言語が使えるのです。

• BASIC プログラム



• BASIC インタプリタ



マシン語モニタと アセンブラ

第
2
章



2.1 マシン語モニタとアセンブラ

この本では、みなさんにマシン語を説明するだけでなく、実際にマシン語というものに触れてもらうために、「マシン語モニタ」と「アセンブラ」を用意しました。

マシン語モニタはメモリ内容の表示やマシン語データの入力、マシン語の実行などをするためのプログラムで、アセンブラはアセンブリ言語で書かれたプログラムを、マシン語のプログラムに変換するためのプログラムです。どちらも、マシン語プログラムの開発にはかかせないものです。本章では、このマシン語モニタとアセンブラの入力の仕方、使い方などを説明しましょう。

なお、本文中に「モニタアセンブラ」とある場合は、マシン語モニタとアセンブラの両方を指します。

2.2 マシン語モニタとアセンブラを入力する前の注意

掲載したモニタアセンブラは（株）MIAより発売されていた「モニタアセンブラ」とほぼ同じものです。現在これを使っている人は、まったく入力する必要がありません。

それ以外のマシン語モニタやアセンブラをすでに持っている人もいるでしょうが、使用方法や機能などが違うので、入力した方がいいかと思います。

モニタアセンブラは16K以上のRAMを持つMSX（MSX、MSX2のどちらでもよい）で使うことができますが、ディスクドライブが繋がっている場合は、動かすことができません。もしディスクが繋がっているときは、ディスク本体をMSXからはずすか、または[SHIFT]キーを押しながらリセットして、ディスクが接続されていない状態にしてから使用してください。なお、当然のことながら、セーブはカセットにしかできません（クイックディスクはだ

いじょうぶですが、セーブ、ロードのコマンドがカセットとは違うので注意してください。

2.3 マシン語モニタの入力

マシン語モニタとアセンブラは、別々に入力します。みなさんの中には、MSX マガジンなどに載っていた他のマシン語モニタを持っている人もいるかもしれませんが、この本のアセンブラを使う場合は、必ずこのマシン語モニタが必要となります。

初めにマシン語モニタを入力します。プログラムリストが P146 にあるので BASIC で入力してください。入力し終わったら必ずセーブすること。このプログラムの 1270 行から 3680 行はマシン語データですから、1 か所でもまちがえてしまうと、実行したときに「暴走」してしまうかもしれません。

セーブは次のようにしてください (☐は リターン キーを押すという意味です)。

CSAVE "MSXMON" ☐

では、実行してみましょう。

RUN ☐

もし「XXXXギョウフキンニ ニュウリョクミスガアリマス」と出てしまったら、その行だけでなく、その前後 2、3 行もいっしょに調べてください。また「Out of DATA」や「Syntax error」、「Illegal function call」、「Type mismatch」などの場合は、画面に表示されている行を中心に、プログラム全体を見直し、DATA 文などがまちがっていないかどうかよく調べてください。

暴走してしまった場合（反応がない、リセットがかかったなど）や動作がおかしいときは、リセットし、もう一度プログラムをロードしてからプログラム全体を見直してください。

2.4 マシン語モニタの使い方

マシン語モニタが正常に動きだした場合には、次のような文字が画面に出てきます。

MSX Monitor Rev1. 1

*■

“*”は「コマンド待ち」の印です。“■”はおなじみのカーソルです。

このマシン語モニタでは、以下に示すコマンドを使うことができます。コマンドの詳しい説明は後で行います。

<コマンド一覧表>

- B : BASICに戻る
- D : メモリの内容を表示する
- S : メモリの内容を変更する
- X : レジスタの内容の表示と変更をする
- G : マシン語プログラムを実行する
- R : マシン語を VRAM からメイン RAM にロードする

これらのコマンドを入力するときに気を付けてもらいたいことは、まちがえて入力してしまったときに、カーソルキーを使って直せないことです。

ではどうするかというと、まず **[BS]** キーでまちがえたところまでカーソルを戻し、そこからもう一度入れ直します。

① * DOOSO, OOFF ■

あ！ まちがえてる。“S” じゃなくて “O” だ

② * DOO ■

BS キーを 7 回押して、カーソルを戻す

③ * D0000, 00FF ■

正しく入れ直して、リターン キーを押す

もし、まちがっているのに リターン キーを押してしまったときは、もう一度初めから打ち直さなければいけません。

① * D00S0, 00FF ↵

あらら！ まちがえてるのに実行しちゃった

② ? 00S0

入力がおかしいですよというメッセージ

③ * D0000, 00FF ■

こんなときは、もう一度初めから入れ直してリターンしてね

2.5 マシン語モニタのコマンドの説明

● B コマンド

書き方

B ↵

マシン語モニタから BASIC に戻るためのコマンドです。

BASIC からマシン語モニタを実行する場合、初めは“RUN”としましたが、2 回目からは“CMD ↵”と打つことによりマシン語モニタになります。ただし、アセンブラといっしょに使っているときは“CMD MON ↵”と打ってください。

1 度 BASIC で入力したマシン語モニタプログラムを実行すると、BASIC にマシン語モニタが組み込まれますから、BASIC のプログラムは消してしまってもかまいません。

B コマンドの例

RUN

MSX Monitor Rev 1.1

*B

OK

CMD

MSX Monitor Rev 1.1

*B

OK


● D コマンド

書き方

D [開始アドレス] [, 終了アドレス] 

DS [開始アドレス] [, 終了アドレス] 

LD [開始アドレス] [, 終了アドレス] 

LDS [開始アドレス] [, 終了アドレス] 

メモリの内容を表示するためのコマンドです。

「開始アドレス」と「終了アドレス」は16進数4ケタで指定します。そのとき"&H8000"や"8000H"のように"&H"や"H"を付ける必要はありません。

終了アドレスをつけなかった場合、128バイト分のメモリの内容が画面に出てきます。

途中で **リターン** キーを押すとコマンドモードに戻り、その他のキーを押すと画面表示が止まります。

「D コマンド」はメモリの内容を16進数とキャラクタで表示し、「DS コマンド」はマシン語チェックサム付きの16進数で表示します(このマシン語チェックサムの計算方法は、横8バイトをすべて足して、その下位2バイトをとったものです。これはMSXマガジンで採用されている方法とは異なりますので注意してください(P46参照)。

頭に“L”が付いているコマンド(LD コマンド、LDS コマンド)は、メモリの内容をプリンタに出力するためのものです。プリンタがない場合は使わないでください(もし、やってしまった場合は **Ctrl** キーと **STOP** キーを同時に押せばコマンド待ちに戻ります)。

D コマンドの例

```
*DD000, D02F
D000 02 30 00 00 30 00 11 23 .0..0..*
D008 34 77 02 0A A4 00 00 00 4w...
D010 00 00 00 00 00 00 00 00 .....
D018 00 00 00 00 00 00 00 00 .....
D020 00 00 00 00 00 00 00 00 .....
D028 00 00 00 00 00 00 00 00 .....
*DSD000, D02F
D000 02 30 00 00 30 00 11 23 : 96
D008 34 77 02 0A A4 00 00 00 : 5B
D010 00 00 00 00 00 00 00 00 : 00
D018 00 00 00 00 00 00 00 00 : 00
D020 00 00 00 00 00 00 00 00 : 00
D028 00 00 00 00 00 00 00 00 : 00
*
```

● S コマンド


書き方

S (アドレス) 

メモリの内容を変更するためのコマンドです。

アドレスは D コマンドと同じように 16 進数で入力します。

このコマンドを行うと、画面にはアドレスと、そのメモリの現在の内容が表示されますので、変更したいときは 16 進数 2 ケタの数を入れて **リターン** キーを押します。もし変更したくないときは、何もせずに **リターン** キーを押します。

もし、まちがえて入力してしまった場合、“**^**  ”でひとつ前のアドレスに戻ります。

コマンド待ちに戻るには “**.**  ” と入力します。

S コマンドの例

```
*SD000
D000 00 11
D001 20 22
D002 03 33
D003 00 55
D004 00 ^
D003 55 44
D004 00 55
D005 00 66
D006 00 .
*SD005
D005 66
D006 00 77
D007 00 .
*
```



● X コマンド

書き方

X 

X (レジスタ名) 

レジスタの内容の表示、変更を行います。

レジスタの内容を表示したいときは、“X” のみ入力し リターン キーを押します

レジスタの内容を変更したいときは、“X” の次に変更したいレジスタ名を続けます。レジスタ名には以下の名前を指定することができます。

<8ビットレジスタ>

A、F、B、C、D、E、H、L

<16ビットレジスタ>

AF、BC、DE、HL、IX、IY、SP、PC

リターン キーを押すと、指定したレジスタの現在の値を表示するので、変更したい場合はそこで値を入力します。値は、8ビットレジスタの場合 16 進数 2 ケタ、16ビットレジスタの場合 16 進

数 4 ケタとなります。

入力して リターン キーを押すとレジスタの内容を書きかえる準備がされます。ここで G コマンドを行うと、レジスタを変更した後マシン語プログラムが実行されます。

X コマンドの例

```
*X
                                SZ H PNC
A =7C    F =37(00110111)
BC=7C34 DE=39DB HL=D3FF
IX=6678 IY=0314 SP=C6E1 PC=EB00
```

```
*XA
A =7C 00
*XBC
BC=7C34 1122
*X
```

```
                                SZ H PNC
A =00    F =37(00110111)
BC=1122 DE=39DB HL=D3FF
IX=6678 IY=0314 SP=C6E1 PC=EB00
```

*

● G コマンド

書き方

G [実行アドレス] [, 停止アドレス 1] [, 停止アドレス 2] 

指定したアドレスから実行します。指定したアドレスにプログラムがない場合、暴走する危険があるので、このコマンドを使うときは十分注意しましょう。

「停止アドレス」を指定すると、そのアドレスまでマシン語が実行された場合、モニタのコマンド待ちに戻ります。このときレジスタの状態が表示されるので、デバッグのときに使うと便利です。

ただし停止アドレスは ROM 上のマシン語プログラムには使えません（つまり 0000～7FFF までは指定できない）。

G コマンドの例

*GD000, D005, D00A

*GD000

● R コマンド

書き方

R 

このコマンドを使うと、VRAM 上に作られたマシン語プログラムがメイン RAM 上にロードされます。

この本に掲載されているアセンブラは、アセンブル（マシン語に変換すること）してできたマシン語プログラムを VRAM 上に作ります（VRAM は、一般に画面表示に使用されますが、必ずしも VRAM 全部が使われているわけではなく、空いているところもたくさんあります。このアセンブラは、そこにマシン語を作りだすようになっています）。ですからそのマシン語を実行するためには、VRAM からメイン RAM にマシン語プログラムをロードしなければなりません。そこで、このコマンドが必要となるわけです。

初めての人には、「アセンブラを使ってできたマシン語プログラムを実行するためには、必ずこのコマンドを一度使わなければならない」とだけ覚えておいてください。

うまくメイン RAM 上にロードできると、このようなメッセージが出てきます。

R コマンドの例 1

*R
Free (C800-D3FF)

Load D000-D006
*

“Free” の後の 16 進数は、ロード可能なマシン語エリアのアドレスで、“Load”の後の 16 進数は、実際にマシン語プログラムがロードされたアドレスです。

うまくロードできなかったときは、このようなメッセージが出ます。

R コマンドの例 2

```
*R
Free ( C800-D3FF )



Load C000
C000 Load error
*
```

“Load error” の前にある 16 進数が、ロードしようとして失敗したアドレスです。


マシン語プログラムは、マシン語エリアの中にロードしなければいけません。マシン語エリアの先頭アドレスは、BASIC の「CLEAR 文」で変更することができ、マシン語が実際にロードされるアドレスは、アセンブリ言語の「ORG 命令」で変更することができます。

2.6 アセンブラの入力

マシン語モニタの使い方を覚えたところで、次はアセンブラを入力しましょう。入力する前には必ず

```
NEW 
CLEAR 0, &HD000 
```

を実行してください。

アセンブラは、モニタの S コマンドを使って入力していきます。プログラムが P151 からあるので、 2-1 のような感じで D400H 番地から EAFFH 番地まで、まちがいなく入力していきましょう。


```

MSX Monitor Rev 1.1
*SD400
D400 00 C3 ←
D401 00 2B ←
D402 00 D4 ←
D403 00 C3
D404 00 18
D405 D4 D4
D406 C3 .
*
D400 C3 2B D4 C3 18 D4 C3 E5 : 19
D408 D5 C3 2E D6 C3 96 D8 C3 : 90
D410 8C D8 C3 25 D4 C3 25 D4 : DC
D418 F3 01 05 00 11 0D FE 21 : 36
D420 26 D4 ED B0 FB C9 C3 00 : 1E

```

図2-1 ダンプリストの入力のしかた

入力したら、DS コマンドで正しく入力できているかどうかを確認し、もし入力ミスがあった場合は S コマンドで直します。なければ BASIC に戻ってマシン語セーブします。このとき、モニタとアセンブラを両方ともセーブして、次からモニタアセンブラとして使えるようにします。

```

*DSD400
D400 C3 2B D4 C3 18 D4 C3 E5 : 19
D408 D5 C3 2E D6 C3 96 D8 C3 : 90
D410 8C D8 C3 25 D4 C3 25 D4 : DC
D418 F3 01 05 00 11 0D FE 21 : 36

```

```

BSAVE "MONASM", &HD400, &HF28F, &HD403

```

以後、モニタアセンブラを使いたいときは、このような順番で行ってください。

- ① MSX の電源を入れる（ディスクは切り離す）
- ② 以下のプログラムを実行する。

```

SCREEN 0:WIDTH 40
CLEAR 0,&HD000
MAXFILES=0

```

- ③ モニタアセンブラをロードし実行する。

```

BLOAD "CAS:",R

```


なお、後々のために、モニタアセンブラ使用時のメモリマップを
図 2-2 に載せておきます。

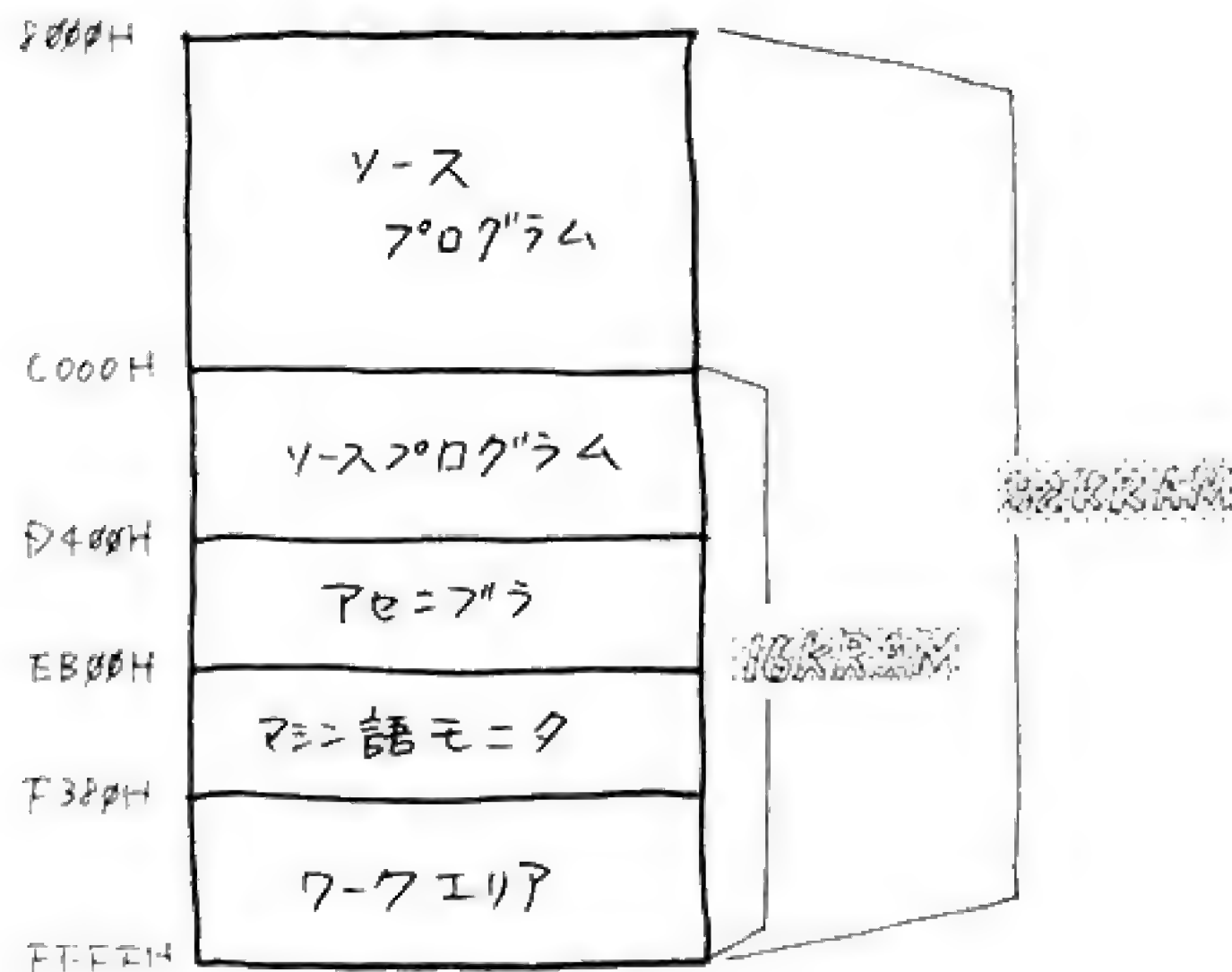


図2-2 モニタアセンブラ使用時のメモリマップ

2.7 アセンブラの使い方

アセンブラは、アセンブリ言語で書かれたプログラムを、マシン語に変換するためのプログラムです。だから、あらかじめアセンブリ言語で書かれたプログラムが入力されていなければ、マシン語は作られません。そこで、初めにプログラムの入力のしかたを説明します。

プログラムの入力、BASIC の REM 文の省略形であるシングルクォートの形で入れていきます。では、ためにリスト 2-1 のプログラムを入力してみてください（このプログラムは RUN しても、ただのコメントなので何もせずに終わってしまいます）。

この本では、プログラムをすべてアルファベットの大文字で書いていますが、小文字でもかまいません。スペースは 1 文字以上あいていれば、いくつでもいいようになっています。なお、入力時に **TAB** キーを使うと便利です（図 2-3）。

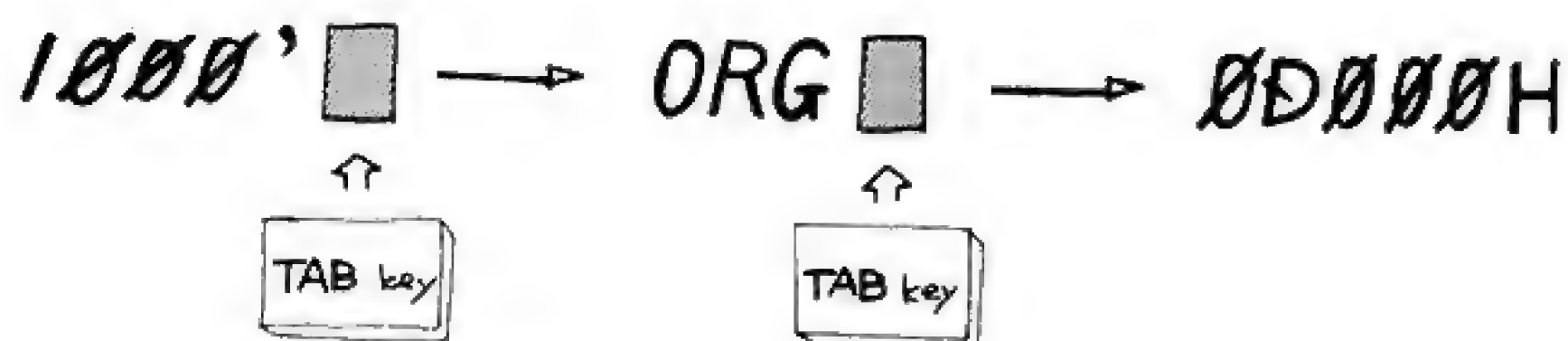


図2-3 TABキーの使い方

アセンブリ言語のプログラムに REM 文のようにコメント（注意書き）を書く場合は、そのかわりに “;（セミコロン）” を使います。すると、これより後はアセンブルされません。

BASIC は英語に近いので、他の人が書いたプログラムでも割と簡単に解読することができますが、アセンブラの場合は自分で書いたプログラムすら、しばらくすると何をしているのかわからなくなってしまうこともあります。自分でプログラムを作ったときには、コメントをなるべく書き入れるようにしましょう。

リスト2-1 プログラム例

```

100 ' ;レジスタ クリアー プログラム
110 '      ORG      0D000H
120 ' MAIN:
130 '      LD      A,0
140 '      LD      B,0
150 '      RET

```

コメント

; A=0
; B=0

次にアセンブルを行います。BASIC から直接次のコマンドを入力してください。

CMD ASM 

MSX Self Assembler Rev 1.1

Pass 1
No Error(s)

Pass 2

```

;レジスタ クリア- 7*07*ラム
D000      ORG      0D000H

```



```

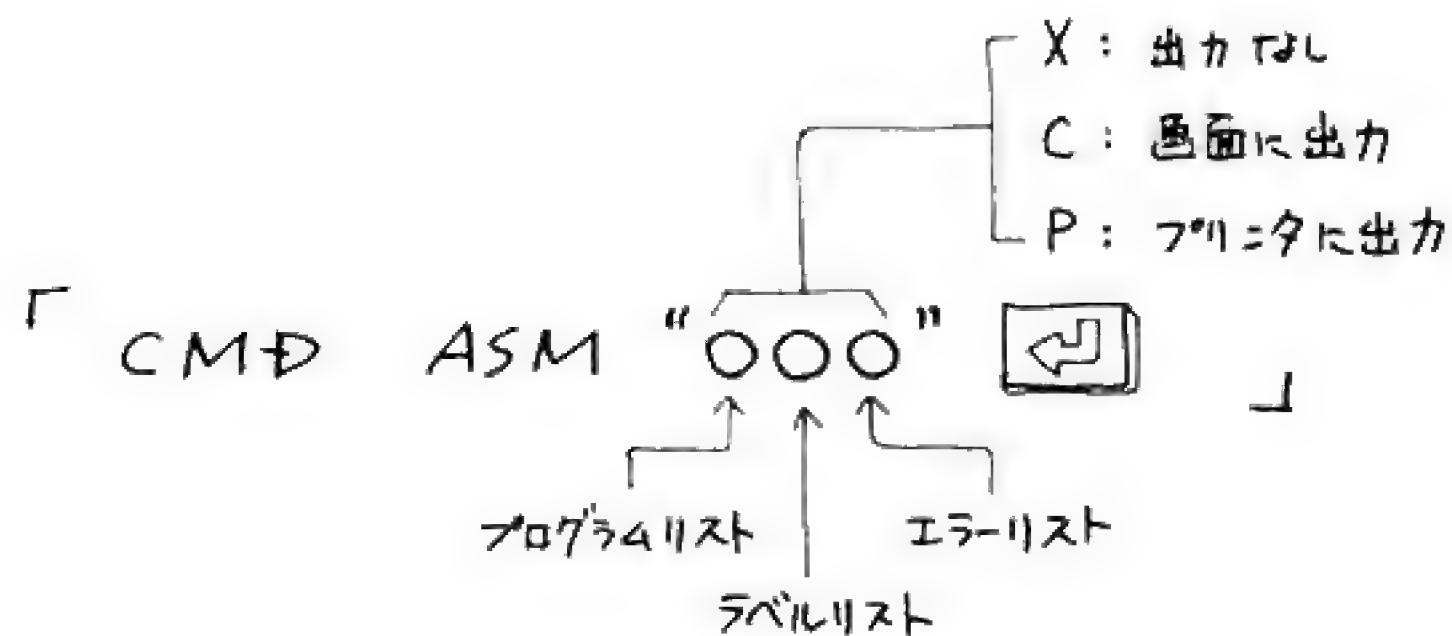
D000          MAIN:
D000 3E00      LD      A,0      ; A=0
D002 0600      LD      B,0      ; B=0
D004 C9        RET
No Error(s)

End Address D005 ,      1 Label(s)

D000  MAIN
Ok

```

画面に出てきたリストを「アセンブルリスト」といいます。アセンブルリストは「プログラムリスト」、「ラベルリスト」、「エラーリスト」から成り、このアセンブラでは図2-4のようにコマンドを書くことにより、これらの出力を制御することもできます。



《例》 プログラムリストをプリンタに出力する。

「CMD ASM "PPP"」

図2-4 CMD ASMのリスト制御

アセンブリ言語のプログラムが、マシン語に変換されました。しかし、このままではマシン語プログラムを実行することはできません。それはP36でも触れたように、今現在はマシン語がVRAMにできているので、これをメインRAMにロードしなければならないからです。これを行うには、モニタのRコマンドを使います。

```
CMD MON
```

```
MSX Monitor Rev 1.1
```

```
*R
```

```
Free ( C800-D3FF )
```

```
Load D000-D005
```

```
*
```

これでマシン語がメイン RAM にロードされました。では本当にロードできているのかどうか「D コマンド」でメモリの中を見てみましょう。

```
*DD000, D02F
```

```
D000 3E 00 06 00 C9 00 00 00 >...ノ...
```

```
D008 00 00 00 00 00 00 00 00 .....
```

```
D010 00 00 00 00 00 00 00 00 .....
```

```
D018 00 00 00 00 00 00 00 00 .....
```

```
D020 00 00 00 00 00 00 00 00 .....
```

```
D028 00 00 00 00 00 00 00 00 .....
```

```
*
```

マシン語プログラムを実行するには、BASIC から行う方法と、マシン語モニタから行う方法の 2 通りがあります。マシン語モニタから実行するには、G コマンドを使いますが、BASIC から実行する場合には次のようにします。

```
*B
```

```
Ok
```

```
DEFUSR=&HD000 .....マシン語の実行アドレスを設定
```

```
Ok
```

```
A=USR(0) .....マシン語を実行
```

```
Ok
```

マシン語プログラムをカセットにセーブ、ロードするには次のようにします。

① カセットにセーブ

```
BSAVE "TEST", &HD000, &HD005, &HD000
```

② カセットからロード

```
CLEAR 0, &HD000  
BLOAD "CAS:"
```

これでひととおり、アセンブラの使い方がわかりました。

2.8 おっと！アセンブルエラーが出ってしまった

アセンブル言語のプログラムに誤りがあった場合には、エラーメッセージが表示されます。エラーメッセージ一覧を次に示します。

☐ Object area full

マシン語をいれるところがいっぱいになってしまった

☐ Label table full

ラベルをいれるところがいっぱいになってしまった

☐ Assembler source error

アセンブラのプログラムでない

☐ Screen not 40×24 text mode

SCREENO でない

☐ Address Overflow

アドレスが OFFFFFH をこえた

☐ Balance Error

カッコ、シングルクォートの使い方がおかしい

☐ Expression Error

式がおかしい

☐ Format Error

プログラムの書き方がおかしい

- [L] Label Error**
ラベル名がおかしい
- [M] Multiply Defined Label**
ラベルが2度定義されている
- [O] Operand Error**
オペランドがおかしい
- [P] Phase Error**
ラベルの値がおかしい
- [R] Reference Error**
相対的な計算がおかしい
- [U] Undefined Label**
ラベルが定義されていない
- [V] Value Error**
値がおかしい

よくありがちなエラーの原因をあげておきますので、もしエラーが出た場合は、このあたりを中心に見直してください。

① アセンブラの書き方に誤りはないか

- 行番号の後ろにシングルクォートはあるか
- コメントの頭にセミコロンはあるか

② 数値の書き方に誤りはないか

- 16進数の頭は数字になっているか
- 文字はシングルクォートに囲まれているか

③ 命令の使い方や書き方に誤りはないか

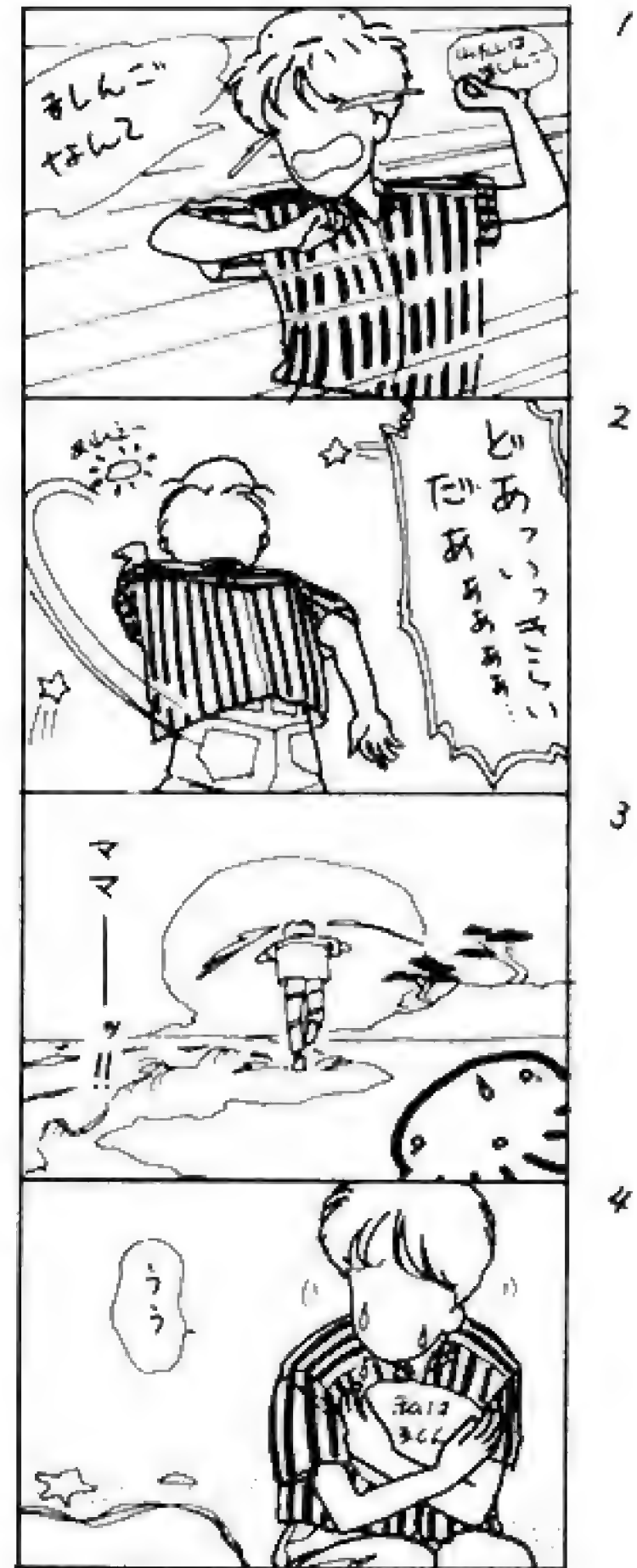
- その命令は本当にあるのか
- オペランドは正しいか

④ ラベルに誤りはないか

- 英文字で始まっているか
- 同じ名前のラベルがないか

2.8 おっと！アセンブルエラーが出しまった

エラーを直す場合は、必ず LIST してから行ってください。アセンブルリストでは直せません。また、MSX2 の場合 WIDTH を 41 以上で表示させていると、アセンブルできません。注意してください。



入力ミスは許さない。チェックサムって何だろう

マシン語のプログラムを入力したとき、ミスをなくすように付けられたチェック用データをチェックサムといいます。チェックサムにはいろいろな方法があるので、あげてみましょう。

① MSX マガジンで採用されている方法

D000	01	01	0A	00	00	00	00	00	00	:DC
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↑
$D0+00+01+01+0A+00+00+00+00+00+00 = DC$										

② この本のマシン語モニタで採用している方法

D000	01	01	0A	00	00	00	00	00	00	:0C
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↑
$01+01+0A+00+00+00+00+00+00+00 = 0C$										

③ 俗にいう縦横チェックサム (②の方法で縦横に計算する)

D000	01	01	0A	00	00	00	00	00	00	:0C	・	0C	
D008	11	22	00	00	00	00	00	00	00	33	:66	→	66
D010	01	01	00	00	00	00	00	00	00	00	:02	→	02
D018	00	00	00	00	00	00	00	00	00	FF	:FF	→	+FF

13 24 0A 00 00 00 00 00 00 32 :73 ⇨ 173

これらのチェックサムの盲点は、算出を数値の合計だけで行っているため、入力ミスしたときでも、たまたまチェックサムの値がリストと合ってしまったら、チェックできないことです。

②の方法のチェックサムで、値が同じになる例

D000	00	00	09	01	02	00	00	00	00	:0C
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↑
$00+00+09+01+02+00+00+00+00+00 = 0C$										

チェックサムが合っていたのに、正常に動作しない場合は、必ず 1 バイト、1 バイトのチェックも忘れてはいけません。

マシン語の 基本命令を 覚えよう



3.1 代入してみよう

BASIC では常識事の代入ですが、マシン語にとってもこれは同じです。マシン語の最初として、この代入からみてみましょう。

代入とは BASIC 言語でいえば「ある変数の内容を特定の値にする」ということです。では、BASIC 言語とアセンブリ言語とでは、どのような違いがあるのでしょうか。

① BASIC 言語での代入

書式 <代入される変数>=<代入する値>

例) A=10
 A=B

② アセンブリ言語での代入

書式 LD <代入されるレジスタ>, <代入する値>

例) LD A, 10
 LD A, B

アセンブリ言語では、代入する命令ということで“LD(ロード)”という二ーモニツクが付きます。しかし、代入する側を右に、される側を左に書くという点では変わりありません。

ここでの例は、変数名とレジスタ名を同じ“A”と“B”で表しましたが、レジスタ名には A、B、C、D、E、H、L しか指定できない、レジスタには 0~255(00H~FFH)までの値しか入れることができない、などの制約があります。LD 命令の詳しい説明は後で行います。

● アセンブリ言語の文の形

BASIC では、1 つの文が「ステートメント」という単位で成り立っていたのに対し、アセンブリ言語では「命令」という単位で成り立っています。アセンブリ言語のプログラムは、この「命令」を並べることによって作られます。「命令」の書式を、次に示します。

＜命令の書式の種類＞

- ① 主命令
- ② 主命令 第 1 オペランド
- ③ 主命令 第 1 オペランド, 第 2 オペランド

「主命令(オペコード)」には、「実行したい操作」が入り、「第 1、第 2 オペランド」には、「その操作の対象となるレジスタ、メモリ、数値」が入ります。「命令」には、このように 3 通りの書き方があり、「主命令」が何であるかによってどの書き方をするか決まっています。実際には P140 のマシン語命令表にまとめてあるので、見ておくといいでしょう。

ちなみに「LD 命令」は、③の書式で書きます。

主命令 第 1 オペランド, 第 2 オペランド
LD A , 2

● アセンブリ言語と BASIC の書き方の違い

アセンブリ言語と BASIC のプログラムとを見くらべてみると次のような違いがあります。

- ① BASIC ではマルチステートメント(1 行に 2 つ以上の文を書くこと)ができるが、アセンブラでは 1 行に 1 つの命令しか書くことができない。
- ② BASIC では、行番号がプログラムを作る上で、なくてはならないものだが、アセンブラでは必要ない。MSX では BASIC のプログラムとしてしか入力できないので、このアセンブラではやむなく行番号をつけているだけ。普通のアセンブラに行番号はない。
- ③ BASIC ではコメントは REM 文に書くが、アセンブリ言語のプログラムでは、“; (セミコロン)” の後ろに書く。
- ④ アセンブリ言語のプログラムは、そのままでは実行できない。

このように、アセンブリ言語と BASIC 言語はまったく違う言語

なので、初めのうちはとまどうかも知れません。毎日、少しずつでも勉強し、なれるようにしましょうね。

● LD (ロード) 命令

BASIC では、代入される側が変数だけだったため、「変数に値を代入する」と「変数に変数の内容を代入する」のどちらかしかありませんでした。しかしマシン語ではレジスタとメモリが代入の対象となるため、命令には主として以下の 4 種類があります。

① レジスタに数値を代入する

LD r, n $\cdots r \leftarrow n$

値 n を、r レジスタに代入する

② レジスタからレジスタに数値を代入する

LD r, s $\cdots r \leftarrow s$

s レジスタの内容を、r レジスタに代入する

③ メモリから A レジスタに数値を代入する

LD A, (nn) $\cdots A \leftarrow (nn)$

nn 番地の内容を、A レジスタに代入する

④ A レジスタからメモリに数値を代入する

LD (nn), A $\cdots (nn) \leftarrow A$

A レジスタの内容を、nn 番地に代入する

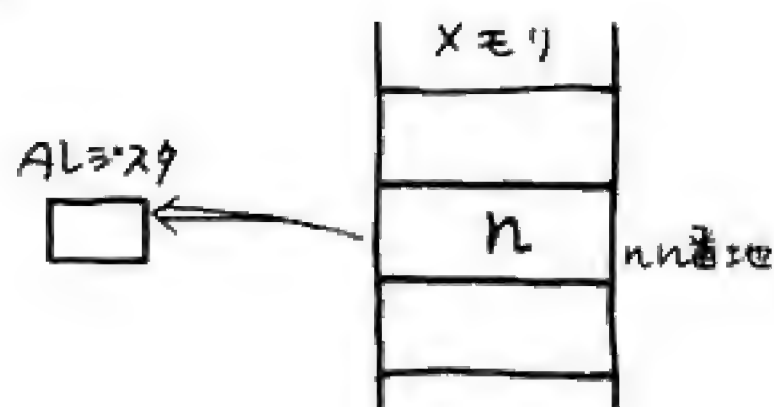
①



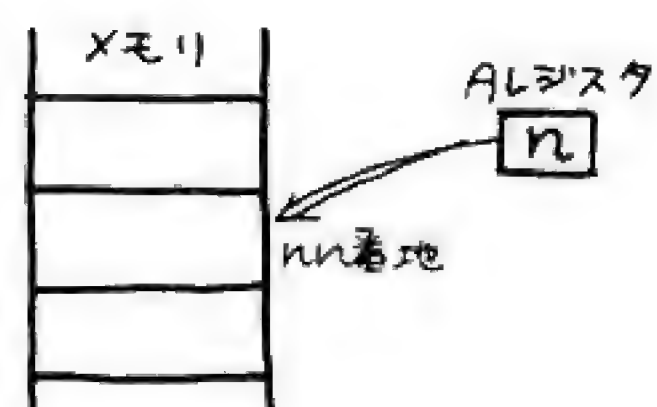
②



③



④



r、sは、それぞれA、B、C、D、E、H、Lレジスタを表します。
また、nは1バイトの数値を表し、nnは2バイトの数値を表します。

③、④ではnnにかッコがついていますが、こうすると「nn番地の内容」つまりnn番地のメモリに入っているデータを指すことになります。もしかッコを忘れてしまうと、「nnという値」となってしまうので、下のようになまったく違う意味になってしまいます。なお、“(nn)”の対象としては、Aレジスタしか使えません。

(2000H) . . . 2000H番地の内容(1バイト)
2000H . . . 2000Hという2バイトの数値

では、ここでLD命令を使ってプログラムを作ってみましょう(リスト3-1)。

リスト3-1 プログラムおよび実行例

```

100 ; **** LDメイレイ 1 ****
110 ;
120 ;     ORG     0D000H
130 ;
140 ;     LD      C,1
150 ;     LD      B,C
160 ;     LD      A,(0D100H)
170 ;     LD      (0D101H),A
180 ;
190 ;     JP      0EB00H
200 ;
210 ;     END

```

```

*GD000
Break EB00

```

```

                SZ H PNC
A =FF      F =00(00000000)
BC=0101  DE=0000  HL=0000
IX=0000  IY=0000  SP=C6E1  PC=EB00

```

```

*
```

このプログラムを実行することにより、C、B、Aレジスタ、そしてメモリに順に数値が代入されていきます。

なお、“JP OEBOOH”という命令は、モニタアセンブラのレジスタ表示ルーチンを実行するためのものです。レジスタの内容はメモリとは違って簡単に見ることができないので、本書ではこのような方法をとりました。こうした場合は、必ずモニタのGコマンドでマシン語を実行してください。

BASIC からUSR 文で、マシン語サブルーチンとして実行したい場合は、ここを“RET”に変更して、アセンブルし直してください（リスト 3-2）。

リスト3-2 プログラムおよび実行例

```
100 ' ;**** LDメイレイ 2 ****
110 '
120 '      ORG      0D000H
130 '
140 '      LD       C,1
150 '      LD       B,C
160 '      LD       A,(0D100H)
170 '      LD       (0D101H),A
180 '
190 '      RET
200 '
210 '      END
```

```
DEF USR=&HD000
Ok
A=USR(0)
Ok
```

“ORG” と “END” については、次に説明します。

● ORG 命令と END 命令

プログラムの初めに “ORG” とあります。これは「ORG(セットオリジン) 命令」といい、本来は「アセンブル時に、いずれのアドレスから実行できるようなマシン語プログラムにするか指定する命令」です。わかりにくいときは図 3-1 ように「アセンブルしてできたマシン語プログラムを、モニタの R コマンドにより、メイン RAM のどのアドレスにロードするか指定」と覚えておいてください。

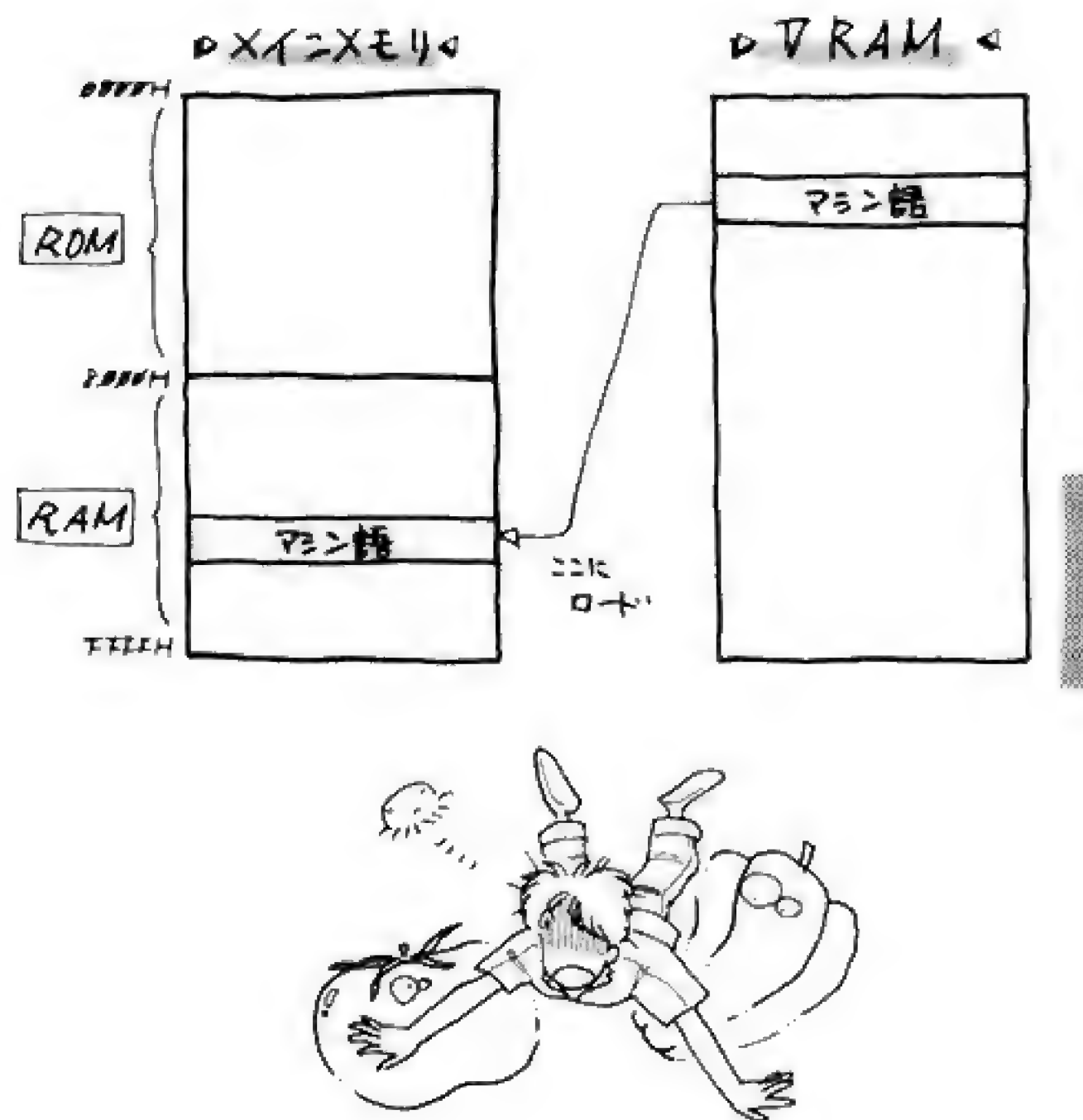


図3-1 ORG命令の機能

プログラムの最後にある“END”は「END (エンド)命令」とい
い、どの行までをアセンブルして、マシン語に変換するかを指定す
るためのものです(図3-2)。BASICのEND文のようにプログ
ラムの実行終了を意味するものではありません。プログラム全部を
アセンブルするときは、これを付ける必要はありません。

100	'	ORG	0D000H	この部分だけ アセンブルされる。
110	'			
120	'	LD	B,C	
130	'	LD	C,02H	
140	'			
150	'	END		
160	'			
170	'	LD	D,E	
180	'	LD	E,04H	

図3-2 END命令の機能

このような、アセンブルするときのための指定を「疑似命令」といいます。これらは××命令という名前がついていますが、あくまでもアセンブラに対し、どのようにアセンブルするかなどを指定するためのもので、これらの命令自身がマシン語になるわけではありません。

● アセンブリ言語での値の表現

このアセンブラでは、10進数、16進数、文字がアセンブリ言語のプログラム中で使えますが、BASICとは書き方が次のように違ってきます。

	BASIC 言語	アセンブリ言語
10 進数	100	100
16 進数	&H64	64H
文字	" A"	'A'

10進数……書き方は、BASICとまったく変わりません。1バイトの記憶場所には0～255、2バイトの記憶場所には0～65535までの値を入れることができます。

16進数……値のあとに"H"を付けます。また、いちばん上のケタがA～Fの場合は、その前に"0"を付けます。これは、よく忘れることがあるので、気を付けてください。

A000H ⇒ 0A000H

FFH ⇒ 0FFH

1バイトの記憶場所には00H～0FFH、2バイトの記憶場所には0000H～0FFFFHまでの値を入れることができます。

文字……文字はシングルフォートで囲みます。1文字は1バイトで表せるので、1バイトの記憶場所には1文字、2バイトの記憶場所には2文字入れることができます。

いずれの場合もアセンブルすると、1 バイトないしは 2 バイトの 2 進数の値となります (リスト 3-3)。

リスト3-3 アセンブル例(プリンタ出力)

100:		; **** LDメイレイ 3 ****	
110:			
120:	D000	ORG	0D000H
130:			
140:	D000 3E64	LD	A, 100
150:	D002 0664	LD	B, 64H
160:	D004 0EB3	LD	C, 0B3H
170:	D006 1641	LD	D, 'A'
180:			
190:	D008 C300EB	JP	0EB00H

● レジスタペアに対する LD 命令

いままで使ってきたレジスタは 8 ビットですから、0~255(00H~FFH) までの数値しか表すことができませんでした。ところが、レジスタには、もう 1 つの使い方として、「2 つのレジスタをつなげてレジスタペア (16 ビットレジスタ) として使う」ことができますから、それにより 0~65535 (0000H~FFFFH) までの数を表すことができるようになります。

2 つのレジスタをつなげるということとは、1 つのレジスタを上位の 8 ビット、もう 1 つを下位の 8 ビットとして図 3-3 のようにつなげて使うことです。この 2 つのレジスタは、どのレジスタでもいいわけではなく、AF、BC、DE、HL のみ可能です。

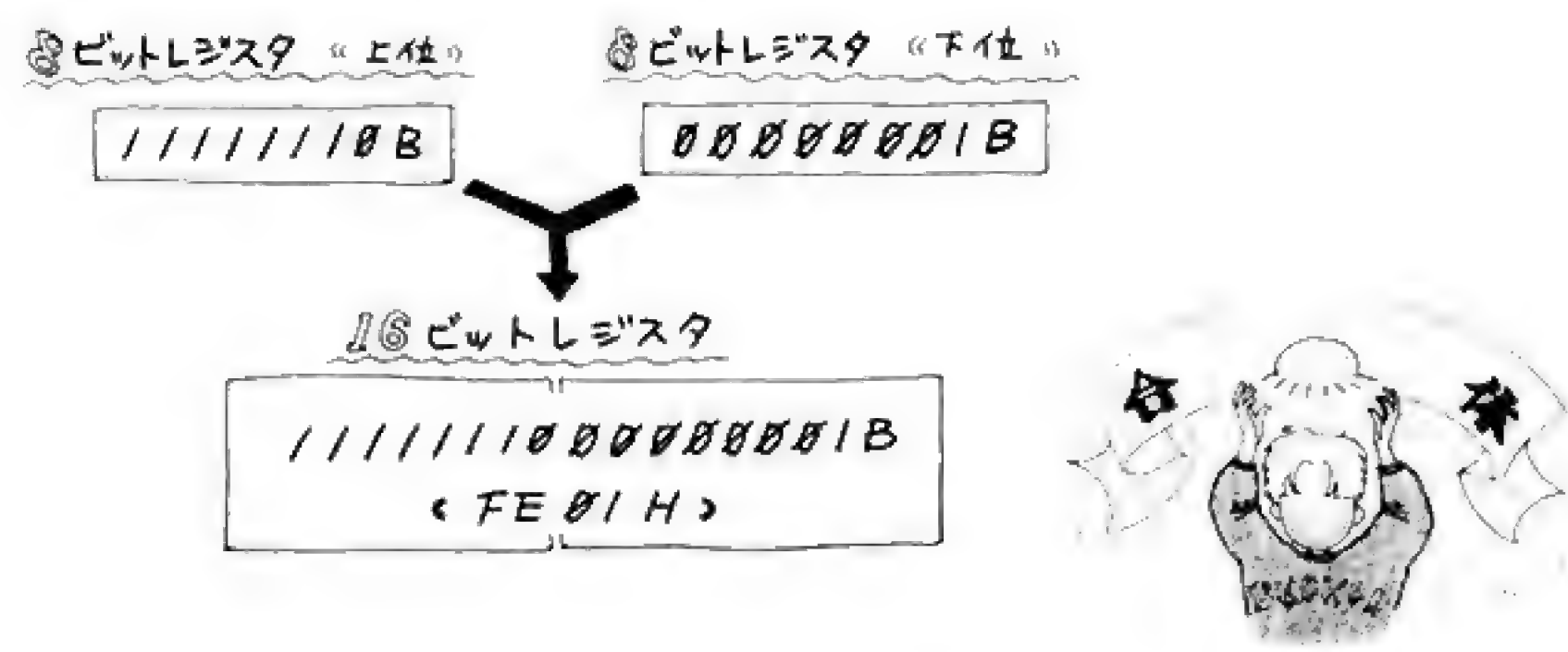


図3-3 16ビットレジスタ

レジスタペアに対するLD命令には次のようなものがあります。

① レジスタに数値を代入する

LD rr, nn ... $rr \leftarrow nn$

2バイトの数値nnを、rrレジスタに代入する

② メモリからレジスタに数値を代入する

LD rr, (nn) ... rrの下位バイト \leftarrow (nn)

rrの上位バイト \leftarrow (nn+1)

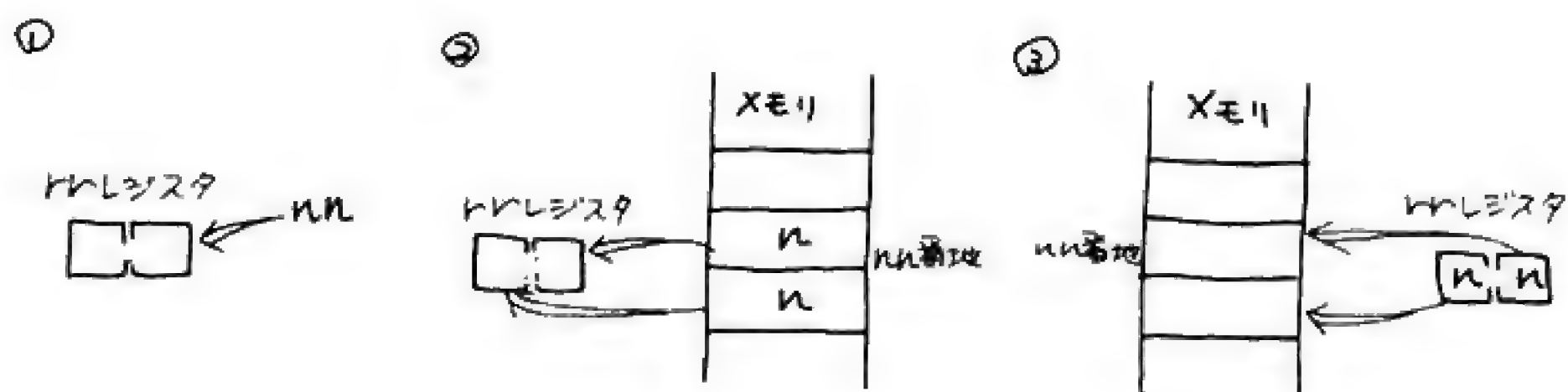
nn番地の内容がrrの下位バイトに、nn+1番地の内容がrrの上位バイトに代入される

③ レジスタからメモリに数値を代入する

LD (nn), rr ... (nn) \leftarrow rrの下位バイト

(nn+1) \leftarrow rrの上位バイト

rrレジスタの下位バイトがnn番地に、rrレジスタの上位バイトがnn+1番地に代入される



rr は、BC、DE、HL レジスタを表します。nn は、2 バイトの数値を表します。レジスタペアの内容をメモリに代入すると、上下バイトが逆になって、メモリに記憶されることに注意してください。

レジスタペアの状態は、レジスタという箱を2 つくっつけたようなものですから、「BCレジスタに 1234H という値が入っている」といったら、Bレジスタには 12H という値、Cレジスタには 34H という値が入っているということになり、もし次に Bレジスタに FFH を代入したとしたら、BCレジスタの値は、FF34H になってしまいます。

この性質は、BASIC の変数にはまるでなかったものです。BASIC では、変数 B と変数 C と変数 BC は、全然別なものですからね。勘違いしないように。

レジスタペアを対象にした LD 命令の使用例を示します (リスト 3-4)。

リスト3-4 プログラム例

```

100 ' ;**** LDメイレイ 4 ****
110 '
120 '      ORG      0D0000H
130 '
140 '      LD       BC, 0AAAAH
150 '      LD       DE, (0D100H)
160 '      LD       (0D102H), BC
170 '
180 '      JP       0EB00H

```

実行すると、BC、DE レジスタ、そしてメモリに数値が代入されます。このときに、マシン語モニタで数値がどのようにメモリに代入されたか確認しておきましょう。

● レジスタペアどうしの代入

レジスタペアに対する代入で、レジスタペアどうしのロード命令がないことに気が付きましたか。別に忘れたわけではありません。実はペアレジスタには、そういう命令がないのです。それでは、具体的に HL レジスタの内容を BC レジスタに代入したいとき、どうすればいいかを考えてみましょう。代表的なやり方としては、次の 3 種類があげられます。

① レジスタペアの上位と下位のレジスタを別々に代入する

```

LD      B, H
LD      C, L

```


- ②レジスタペアの内容をメモリに入れて、別のレジスタペアに代入する

```
LD      (9000H), HL
LD      BC, (9000H)
```

- ③スタックを使う（この命令については、P 61 で説明します）

```
PUSH   HL
POP     BC
```

いちがいにはどの方法がいいとは言えませんが、一般的には③が多く使われるようです。これは趣味の問題です。

これらのプログラム例を、3つの方法について行ってみます（リスト3-5、リスト3-6、リスト3-7）。

リスト3-5 プログラム例

```
100 ' ; **** LDメイレイ 5 ****
110 '
120 '      ORG      0D000H
130 '
140 '      LD       HL, 0A0BH
150 '      LD       B, H
160 '      LD       C, L
170 '
180 '      JP       0EB00H
```

リスト3-6 プログラム例

```
100 ' ; **** LDメイレイ 6 ****
110 ' /
120 '      ORG      0D000H
130 '
140 '      LD       HL, 0A0BH
150 '      LD       (0D100H), HL
160 '      LD       BC, (0D100H)
170 '
180 '      JP       0EB00H
```

リスト3-7 プログラム例

```

100 ' ; **** LDメイレイ 7 ****
110 '
120 '      ORG      0D000H
130 '
140 '      LD       HL, 0A0BH
150 '      PUSH     HL
160 '      POP      BC
170 '
180 '      JP       0EB00H

```

● こんな方法もある、レジスタ間接

8ビットレジスタの代入命令で“LD A, (nn)”という命令を説明しました。これを少し拡張したものとして、レジスタペアを使い、次のようなことができます。

- ① rrレジスタの内容の指すアドレスの内容を、Aレジスタに代入する

LD A, (rr) ... $A \leftarrow (rr)$

- ② Aレジスタの内容を、rrレジスタの内容の指すアドレスに代入する

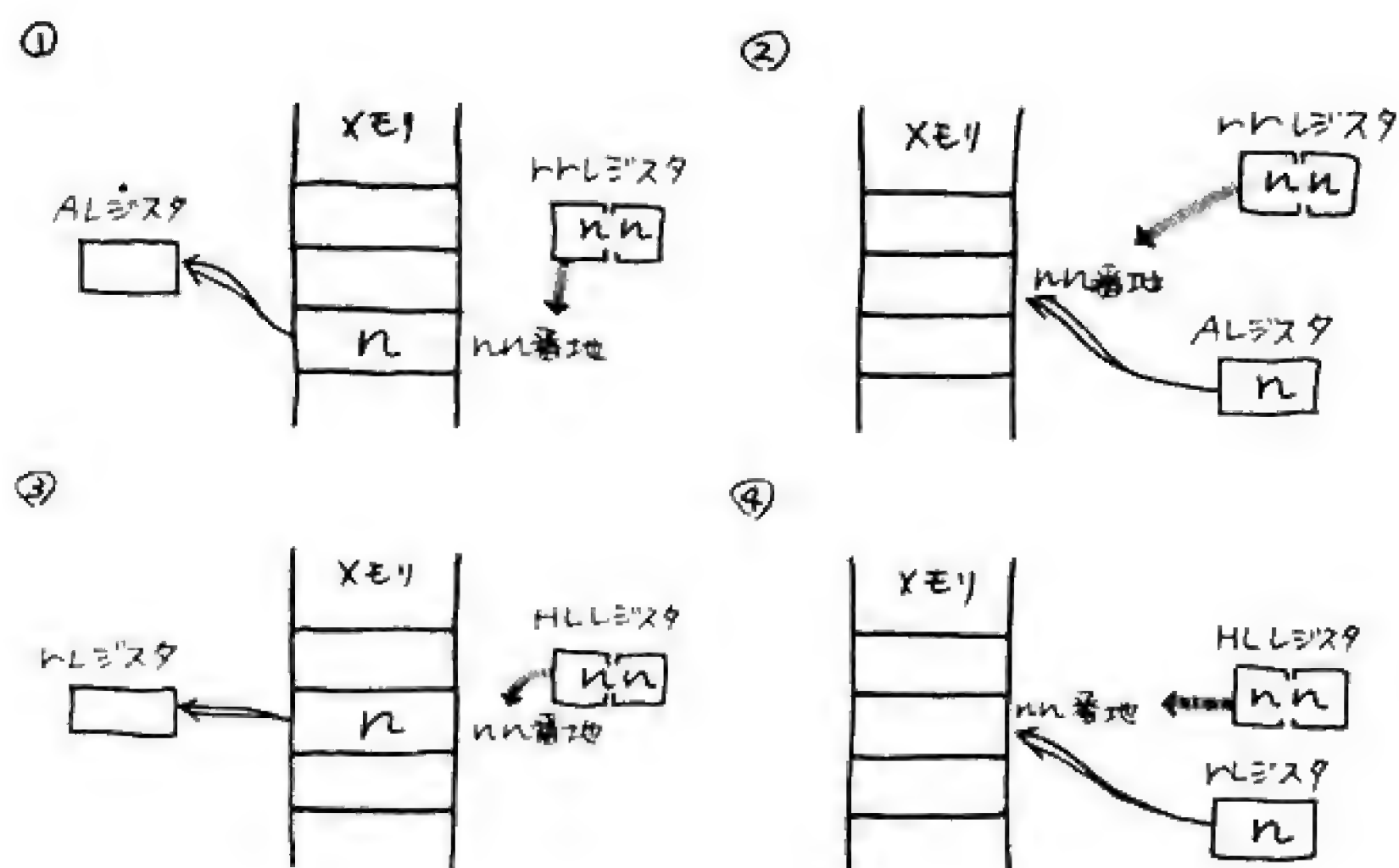
LD (rr), A ... $(rr) \leftarrow A$

- ③ HLレジスタの内容の指すアドレスの内容を、rレジスタに代入する

LD r, (HL) ... $r \leftarrow (HL)$

- ④ rレジスタの内容を、HLレジスタの内容の指すアドレスに代入する

LD (HL), r ... $(HL) \leftarrow r$



nr は、BC、DE レジスタのいずれかを表します。r は、A、B、C、D、E、F レジスタを表します。

nr の内容をアドレスとしてみると“(nr)”はそのアドレスのメモリの内容を示すことになります。今、HL レジスタに D000H という値が入っているとすると、“(HL)”は、D000H 番地のメモリの内容を示すことになり、“(0D000H)”と書いたのと同じことになります。

なお、①、②のように、BC、DE レジスタを使ったときは、必ずもう一方のレジスタが A レジスタでなければなりません。③、④のように、HL レジスタのときは、A レジスタ以外も使うことができます。

レジスタ間接を使ったプログラム例を示します (リスト 3-8)。

リスト3-8 プログラム例

```

100 ' ;**** LDメイレイ 8 ****
110 '
120 '      ORG      0D000H
130 '
140 '      LD       A, 33H
150 '      LD       (0D100H), A
160 '      LD       A, 0
170 '
180 '      LD       HL, 0D100H
190 '      LD       A, (HL)
    
```

200	'		
210	'	LD	BC, 0D102H
220	'	LD	(BC), A
230	'		
240	'	LD	HL, 0D102H
250	'	LD	B, (HL)
260	'		
270	'	LD	HL, 0D104H
280	'	LD	(HL), B
290	'		
300	'	JP	0EB00H

実行すると、D100H 番地、D102H 番地、D104H 番地に数値が代入されていきます。

● スタックに対する命令

マシン語では、大きなプログラムを作っていると、レジスタの内容をいつとき保存しておきたいことがあります。例えば、レジスタペアをすべて使ってしまったときに、HL レジスタと BC レジスタを至急使いたいときなどです。こんな場合、単純に考えると例 1 のようになります。

<例 1> LD (0D100H), HL } レジスタの内容を、適当な
LD (0D102H), BC } アドレスに入れておく
<BC、HL レジスタを使った処理>

LD BC, (0D102H) } レジスタの内容を
LD HL, (0D100H) } もとどおりに戻す

レジスタペアの内容を、一時的にメモリに保存することを「退避」と言い、それをもとのレジスタペアに戻すことを「復帰」といいます。マシン語には、これを行う方法としてスタックを用いる方法があり、それに関連した命令として、PUSH(プッシュ)命令、POP(ポップ)命令があります。

① 退避

PUSH rr ... (SP-1) ← rr の上位バイト
(SP-2) ← rr の下位バイト
SP ← SP-2

rr の内容をスタックに退避する

② 復帰

POP rr ... rr の上位バイト ← (SP)
rr の下位バイト ← (SP+1)
SP ← SP+2

スタックから、内容を rr に復帰する

rr には、BC、DE、HL、AF レジスタを指定することができます。では、例 1 をスタックを使って書き直してみましよう。スタックを使うと、そのためのメモリのアドレスを気にしなくてもいいので、プログラムがスッキリします。

<例 2> PUSH HL } レジスタの内容を、スタック
 PUSH BC } に入れておく
 <BC、HL レジスタを使った処理>

POP BC } 終わったら、レジスタの内容を
POP HL } もとに戻しておく

ここで気を付けることは、いくつかのレジスタをまとめて POP するときに、PUSH したときとは反対の順番でする必要があることです。この場合も、HL → BC の順で PUSH しているので、POP するときは、逆に BC → HL の順に行っています。もし、そのままの順に行うと、BC に HL の内容が、HL に BC の内容が代入されてしまいます。

なお、PUSH したら、その分は必ず POP しないとスタックが破壊されてしまいます。そのわけは後で説明します。

スタックを使ったプログラム例を示します（リスト 3-9）。

リスト3-9 プログラム例

```

100 ' ;**** スタック ****
110 '
120 '      ORG      0D000H
130 '
140 '      LD       HL, 0BBBBH
150 '
160 '      PUSH     AF
170 '      PUSH     HL
180 '      LD       HL, 0AAAAH
190 '      LD       (0D100H), HL
200 '      POP      HL
210 '      POP      AF
220 '
230 '      LD       (0D102H), HL
240 '
250 '      JP       0EB00H

```

実行すると、AF、BC、DE、HL レジスタの内容が順にスタックに退避し、その後、逆順に復帰していきます。

● スタックのしくみ

スタックとは、具体的にいうと SP（スタックポインタ）レジスタが指しているメモリのことです。

PUSH 命令が行われると、そこにレジスタペアの内容 2 バイトが代入されます。次に、SP レジスタから 2 が自動的に引かれ、SP レジスタは新たなメモリを指します。PUSH 命令が実行されるたびに、データがメモリに積まれていくような感じになります。これがスタック（積み重ね）と呼ばれるゆえんです。

逆に POP 命令が行われると、SP が指すアドレスから 2 バイト分のデータがレジスタペアにロードされ、SP に自動的に 2 が加えられます（図 3-4）。なお、このときスタックとして使われるメイン RAM 上のメモリを「スタックエリア(スタック領域)」と呼びます。

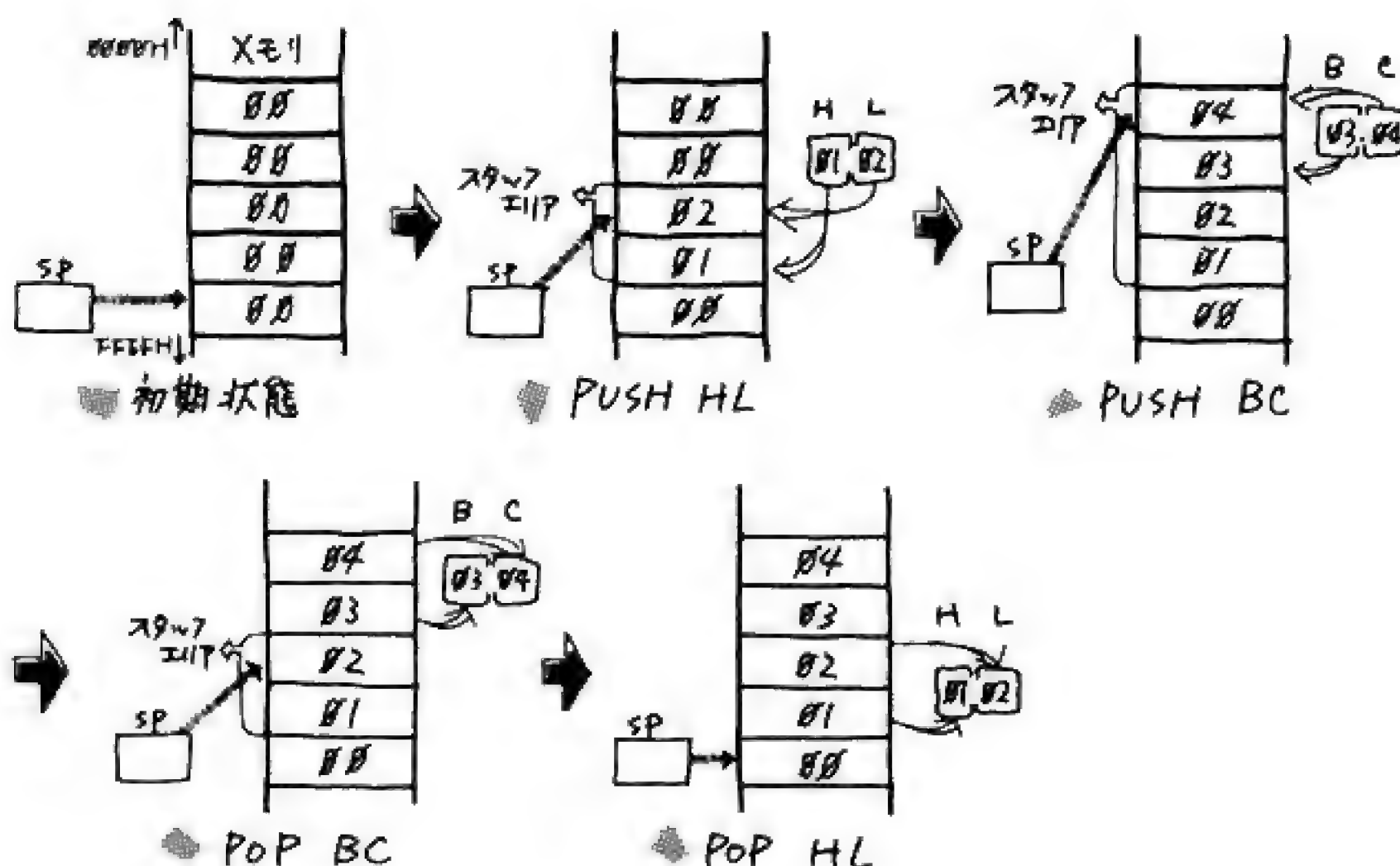


図3-4 スタックの使われ方

スタックエリアを設定するために、SPレジスタを一度は設定しなければなりません。しかし、普通はすでにBASICで設定されているので、プログラムで新たに設定し直す必要はありません。また、設定し直したために、暴走することもありますから、さわらぬ神にたたりなしです。

スタックの使い方を覚えることは、プログラムを作る上で、非常に有用なことです。少し難しいとは思いますが、できるだけ使えるようにしておきたい命令です。

● スタックが破壊される（その1）

スタックが破壊されるということが、どのようなことを説明しましょう。スタックが壊れるということは、プログラムの実行前と実行後でSPレジスタの内容が変わってしまったり、スタックエリアの内容そのものが変化してしまうことをいいます。では、SPレジスタの内容が変わる、例をあげてみましょう。

〈例〉 PUSH HL
 PUSH BC
 PUSH BC

この場合、POP 命令が 1 回少ないので、スタックは初めに比べ 2 してしまいます。もしこのプログラムが、何回も実行されたならば、スタックはどんどんマイナスされ、そこにあったプログラムやデータは、ことごとく破壊されてしまいます。

● ラベル（その 1）

いままでは、アドレスやデータを数値で直接指定しましたが、これを「ラベル」を使うことによって、もっとわかりやすいプログラムに仕立てることができます。たとえば例 1 をラベルを使って書き直すと例 2 のようになります

<例 1>	LD A, (9000H)
	LD (9001H), A
<例 2>	LEFT EQU 9000H
	RIGHT EQU 9001H
	LD A, (LEFT)
	LD (RIGHT), A

“EQU” は、アセンブラの疑似命令の 1 つで、EQU（イクエート）命令と言います。この EQU 命令により、ラベル “LEFT” と “9000H”、ラベル “RIGHT” と “9001H” は等しいものとみなされます。つまり、このプログラムの中には “LEFT” というラベルが使われている場合は、アセンブルするときにその部分を “9000H” だとして、変換されることになります。

ラベルを好みの名前にしておくと、アドレスや数値を文字として覚えることができます。もちろん例 1 と例 2 はアセンブルするとまったく同じマシン語になります（リスト 3-10）。

リスト3-10 例1と例2のアセンブルリスト(プリンタ出力)

```

100:                                ;**** ラベル ****
110:
120:    D000                        ORG        0D000H
130:
140:    9000 =                      LEFT EQU    9000H
150:    9001 =                      RIGHT EQU   9001H
160:
170:    D000 3A0090                  LD         A, (9000H)
180:    D003 320190                  LD         (9001H), A
190:
200:    D006 3A0090                  LD         A, (LEFT)
210:    D009 320190                  LD         (RIGHT), A
220:
230:    D00C C300EB                  JP         0EB00H

9000  LEFT          9001  RIGHT

```

ラベルは、以下の制約を守って付ける必要があります。

- ① アルファベットと数字以外を使ってはいけない
- ② 6文字以内でなければならない
- ③ 初めの文字は、アルファベットである必要がある
- ④ レジスタ名と同じ名前を使ってはいけない

× A、B、C、D、E、F、H、L、I、R
 AF、BC、DE、HL、PC、SP、IX、IY

- ⑤ 命令と同じ名前を使ってはいけない

× LD、PUSH、POP、EQU、END など

- ⑥ 途中にスペースが入ってはいけない

× STA_RT

3.2 計算してみよう

BASIC では、四則演算などを初めとして、いろいろな算術演算子や関数を持っているので、自由自在に計算することができます。ところが、マシン語には基本的に加算（たし算）命令と減算（ひき算）命令しかありません。ちょっと寂しいような気がしますね。しかし、ゲームなどのプログラムなどでは、かけ算やわり算を使うことは実際あまりないので、今のところ必要ないものと考えてください。なお、かけ算やわり算はプログラムで作り出すことが可能です。

① 加算

・BASIC 言語

書式 $\text{〈足された結果〉} = \text{〈足される値〉} + \text{〈足す値〉}$

例) $A = A + 10$
 $A = A + B$

・アセンブリ言語

書式 **ADD** $\text{〈Aレジスタ〉}, \text{〈足す値〉}$

例) **ADD** $A, 10$
 ADD A, B

② 減算

・BASIC 言語

書式 $\text{〈引かれた結果〉} = \text{〈引かれる値〉} - \text{〈引く値〉}$

例) $A = A - 10$
 $A = A - B$

・アセンブリ言語

書式 **SUB** 〈引く値〉

例) **SUB** 10
 SUB B

ADD（アド）命令は加算命令で、SUB（サブトラクト）命令は減算命令です。マシン語の加算および減算命令では、足されたり引かれたりする数値が、必ず A レジスタにあらかじめ入っていないとなりません。また、計算結果は必ず A レジスタに入ります。SUB 命令では、＜引かれる値＞に相当するものを書きませんが、暗黙的に A レジスタが引かれる対象となります。

● 2 進数の加算、減算

ここで、2 進数の加算、減算のやり方を覚えておきましょう。2 進数の加算、減算は基本的に 10 進数のそれと変わりありません。手初めに 2 進数 1 ケタの計算を行ってみます。

① 2 進数 1 ケタの加算

$$\begin{array}{r} 0 \\ + 0 \\ \hline 0 \end{array} \quad \begin{array}{r} 0 \\ + 1 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ + 0 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ + 1 \\ \hline 10 \end{array}$$

② 2 進数 1 ケタの減算

$$\begin{array}{r} 0 \\ - 0 \\ \hline 0 \end{array} \quad \begin{array}{r} 0 \\ - 1 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ - 0 \\ \hline 1 \end{array} \quad \begin{array}{r} 1 \\ - 1 \\ \hline 0 \end{array}$$

次のような、1 バイトどうしの場合はどうでしょうか。これも、縦書き計算で表してみましょう。

$$\textcircled{1} 4\text{DH} + 21\text{H} =$$

$$\textcircled{2} 4\text{DH} - 21\text{H} =$$

この結果は、次のようになります。簡単ですね。

$$\begin{array}{rcl}
 \textcircled{1} & 01001101 & =4\text{DH} \\
 + & 00100001 & =21\text{H} \\
 \hline
 & 01101110 & =6\text{EH}
 \end{array}$$

$$\begin{array}{rcl}
 \textcircled{2} & 01001101 & =4\text{DH} \\
 - & 00100001 & =21\text{H} \\
 \hline
 & 00101100 & =2\text{CH}
 \end{array}$$

● 繰り上がり、繰り下がり

加算、減算命令の結果は、マシン語の場合必ず Aレジスタに入ります。では、もし計算結果が FFH よりも大きくなるか、または 00H よりも小さくなって、Aレジスタに入らないような数値になったらどうなるのでしょうか。

例として、次の計算を行ってみましょう。

$$\textcircled{1} \text{FFH} + 01\text{H} =$$

$$\textcircled{2} 00\text{H} - 01\text{H} =$$

この結果は、あきらかに Aレジスタには入りきりませんね。これを、加算、減算命令を使って計算すると、次のようになります。

$$\begin{array}{rcl}
 \textcircled{1} & 11111111 & =\text{FFH} \\
 + & 00000001 & =01\text{H} \\
 \hline
 & \textcircled{1}00000000 & =00\text{H} \\
 & \uparrow & \\
 & \text{無視される} &
 \end{array}$$

あるものとして考える

↓

$$\begin{array}{rcl}
 \textcircled{2} & \textcircled{1}00000000 & =00\text{H} \\
 - & 00000001 & =01\text{H} \\
 \hline
 & 11111111 & =\text{FFH}
 \end{array}$$

この通り、マシン語の加算、減算を実行した後のレジスタには、繰り上がりや繰り下がりを見捨てた部分が残ることになります。

ところで、この例をもう1度よく見ると、「FFHの次の数は00H」あるいは、「00Hの前の数はFFH」であることに気が付きます。もうすこしワクを広げてみましょう。

・・・FDH、FEH、FFH、00H、01H、02H・・・

こうなるはずで、それらの数の並び全体を表したものが図3-5です。このように考えると、FFHに02Hを足したときとか、02Hから05Hを引いたときの結果がどうなるのか考えやすいですね。

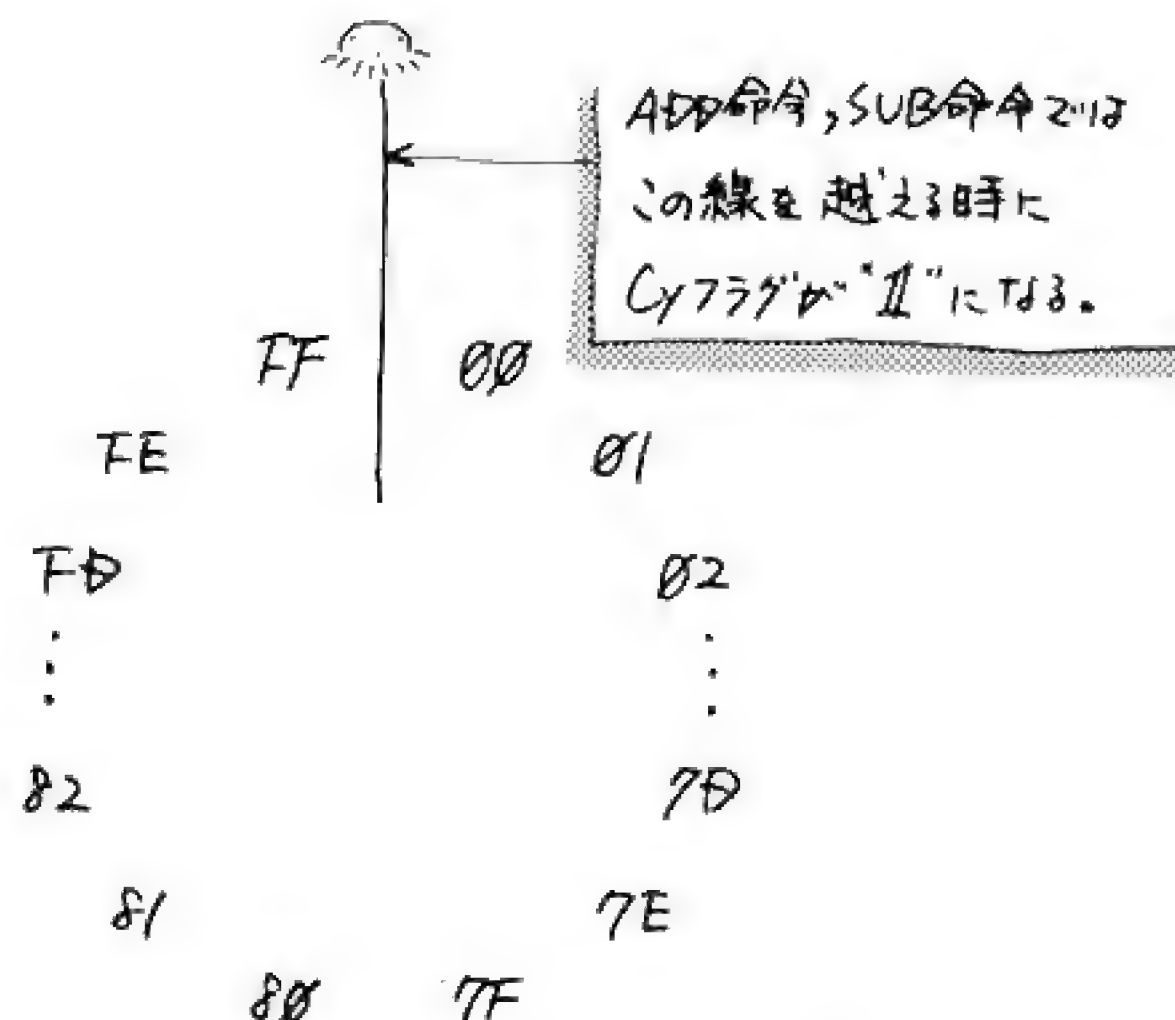


図3-5 数の並びの関係

同様に、2バイトの値では、FFFFHと0000Hがつながっているという見方ができます。

● ADD 命令

ADD (アド) 命令を使うときには、あらかじめAレジスタに足される値を入れておく必要があります。そして、計算の結果は必ずAレジスタに入ります。ADD 命令には以下の3種類があります。

① Aレジスタと数値を足す

ADD A, n ... $A \leftarrow A + n$

値 n と A レジスタの内容を足す。結果は A レジスタに入る。

② Aレジスタとrレジスタを足す

ADD A, r ... $A \leftarrow A + r$

r レジスタの内容と A レジスタの内容を足す。結果は A レジスタに入る。

③ Aレジスタと間接指定されたメモリの内容を足す

ADD A, (HL) ... $A \leftarrow A + (HL)$

HL レジスタで指定されたメモリと A レジスタの内容を足す。
結果は A レジスタに入る。

n は 1 バイトの数値を表します。r は A、B、C、D、E、H、L レジスタを表します。

ADD 命令を使ったプログラム例を示します (リスト 3-11)。

リスト3-11 プログラム例

```

100 ' ; **** ADDメイレイ 1 ****
110 '
120 '      ORG      0D000H
130 '
140 '      LD       A, 2
150 '      ADD      A, 4
160 '      LD       (0D100H), A
170 '
180 '      LD       A, 2
190 '      LD       B, 5
200 '      ADD      A, B
210 '      LD       (0D101H), A
220 '
230 '      LD       A, 2
240 '      LD       HL, 0D101H
250 '      ADD      A, (HL)
260 '      LD       (0D102H), A
270 '
280 '      JP       0EB00H

```

ここで、繰り上がりのある例についても実行してみましょう。Aレジスタに FFH、Bレジスタに 03Hが入っていたとき、これらを対象に ADD 命令を実行するとどうなるでしょうか(リスト 3-12)。

リスト3-12 プログラムおよび実行例

```

100 ' ; **** ADDメイレイ 2 ****
110 '
120 '      ORG      0D000H
130 '
140 '      LD       A, 0FFH
150 '      LD       B, 03H
160 '      ADD      A, B
170 '
180 '      JP       0EB00H

```

```

*GD000
Break EB00

```

```

                        SZ H PNC
A = 02  F = 11 (00010001)
BC=0300 DE=0000 HL=0000
IX=0000 IY=0000 SP=C6E1 PC=EB00

```

*

AレジスタにBレジスタの内容が足され、結果はAレジスタに入ります。この場合は繰り上がりがあるので、前に言ったようにAレジスタには、その部分が無視した部分が入ることになります。ここで大事なことは、実はこの無視された部分がFレジスタのCy(キャリー)フラグに現れることです。つまり、繰り上がりがあった印として、Cyフラグが1になります。また、逆に繰り上がりがない場合は0になります。Cyフラグは、計算のたびに0か1のどちらかに変化しています。

```

  11111110  =FEH ← Aレジスタ
+ 00000011  =03H ← Bレジスタ
-----
  00000001  =01H → Aレジスタ

```

1 → Cyフラグ

● SUB 命令

ADD 命令と同じように、SUB(サブトラクト)命令も A レジスタが計算の対象となりますが、アセンブリ言語では、A レジスタの対象となるものしか書き表さないの注意してください。

① A レジスタから数値を引く

SUB n ... A ← A - n

A レジスタの内容から値 n を引く。結果は A レジスタに入る。

② A レジスタから r レジスタを引く

SUB r ... A ← A - r

A レジスタの内容から r レジスタの内容を引く。結果は A レジスタに入る。

③ A レジスタからメモリの内容を引く

SUB (HL) ... A ← A - (HL)

A レジスタの内容から HL レジスタで指定されたメモリの内容を引く。結果は A レジスタに入る。

n は 1 バイトの数値を表します。r は A、B、C、D、E、H、L レジスタを表します。

SUB 命令を使ったプログラム例を示します (リスト 3-13)。

リスト3-13 プログラム例

```

100 ' ; **** SUBメイレイ 1 ****
110 '
120 '      ORG      0D000H
130 '
140 '      LD       A, 9
150 '      SUB      4
160 '      LD       (0D100H), A
170 '
180 '      LD       A, 9
190 '      LD       B, 3
200 '      SUB      B
210 '      LD       (0D101H), A
220 '

```

```

230 '      LD      A,9
240 '      LD      HL,0D101H
250 '      SUB     (HL)
260 '      LD      (0D102H),A
270 '
280 '      JP      0EB00H

```

ADD 命令と同じように、この命令でも繰り下がりと Cy フラグが 1 になります。A レジスタに 05H、B レジスタに 12H が入っているときに、それらを対象として SUB 命令を実行した場合で考えてみましょう (リスト 3-14)。

リスト3-14 プログラムおよび実行例

```

100 ' ;**** SUBメイレイ 2 ****
110 '
120 '      ORG      0D000H
130 '
140 '      LD      A,05H
150 '      LD      B,12H
160 '      SUB     B
170 '
180 '      JP      0EB00H

```

```

*GD000
Break EB00

```

```

                                SZ H PNC
A =F3    F =A3(10100011)
BC=1200 DE=0000 HL=0000
IX=0000 IY=0000 SP=C6E1 PC=EB00

```

*

```

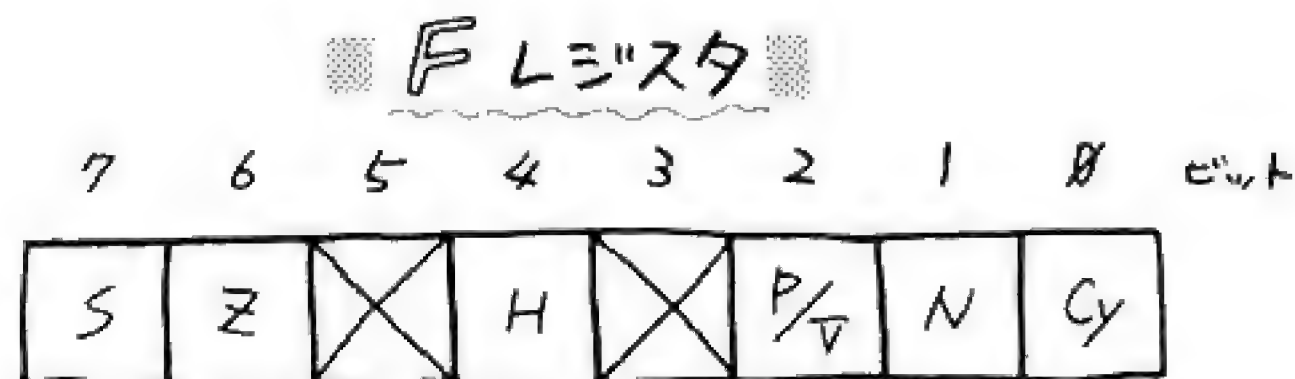
00000101 =05H ← Aレジスタ
- 00010010 =12H ← Bレジスタ
-----
11110011 =F3H → Aレジスタ

```

1 → Cyフラグ

●レジスタの秘蔵っ子、フラグ

ここでフラグの話をしておきましょう。フラグとは、いずれかの命令を実行したとき、その結果に応じて変化するもので、具体的にはFレジスタの各ビットがひとつのフラグとなっています。Fレジスタは図3-6のようになっています。



S : サインフラグ

Z : ゼロフラグ

H : ハーフキャリーフラグ

P/V : パリティ・オーバーフローフラグ

N : 演算フラグ

Cy : キャリーフラグ

× : 使用しない

図3-6 Fレジスタの構成

たとえば、繰り上がりがあったかどうかを示すためのCyフラグは、Fレジスタの0ビット目を指し、繰り上がりがあった場合は、自動的にこのビットが1になります。ただし、命令によっては、どんな場合でもフラグがまったく変わらないものや、特定のフラグのみ変わるものがありますから、新しい命令を覚えるたびに確認してください。フラグが実際、何の役に立つのかは、読み進んでいくうちにわかってくると思います。

フラグを0にすることを「××フラグをリセットする」または「クリアする」といい、1にすることを「××フラグをセットする」または、フラグが「旗」を意味することから「××フラグを立てる」といいます。

フラグは全部で6種類あるわけですが、本書では特に重要なCyフラグとZフラグについてだけ紹介します。

① Cy (キャリー) フラグ

繰り上がり、繰り下がりがあったときはセット。ないときはリセットされる。

② Z (ゼロ) フラグ

演算結果が 0 のときはセット。0 でないときはリセットされる。

LD 命令、PUSH 命令、POP 命令では、いずれのフラグも変化しません。ADD 命令、SUB 命令では、Cy フラグと Z フラグのどちらも変化します。その他の命令については、随時解説します。

● レジスタペアの加算、減算命令

レジスタペアの加算のときは、A レジスタに代わって HL レジスタがその役割を果たします。命令は、次のようになっています。

① 加算

ADD HL, rr ... $HL \leftarrow HL + rr$

HL レジスタの内容と rr レジスタの内容を足して、その結果を HL レジスタに入れる。

② 減算

レジスタペアの一般的な減算命令はありません。しかし、SBC (キャリー付減算) 命令を使うことにより、同じ働きをさせることができます。これについては、SBC 命令の項を見てください。

rr は BC、DE、HL レジスタを表します。フラグは Cy フラグのみ変化します。

この命令は、レジスタペアどうしのみ行うことができます。レジスタペアは大きな数値があつかえるので、便利のように感じますが、一般的に 8 ビットレジスタの方が 16 ビットレジスタよりも機能が豊富であり、レジスタペアに対する命令はオマケのようなものですから、こればかり使うのはひかえるべきです。

レジスタペアの ADD 命令のプログラム例を示します (リスト 3-15)。

リスト3-15 プログラム例

```

100 ' ;**** 16bit ADD ****
110 '
120 '      ORG      0D000H
130 '
140 '      LD       HL,1000H
150 '      LD       BC,1111H
160 '      ADD      HL,BC
170 '
180 '      JP       0EB00H

```

●ADC 命令と SBC 命令

ADD 命令、SUB 命令と似たものに、ADC（アド・ウイズ・キャリー）命令と SBC（サブトラクト・ウイズ・キャリー）命令があります。この命令は、ADD 命令や SUB 命令とほとんど同じですが、唯一異なる点は、第 1 オペランドと第 2 オペランドとの計算の他に、Cy フラグも計算の対象になるということです。

- ① ADC A, n ... $A \leftarrow A + n + Cy$
- ② ADC A, r ... $A \leftarrow A + r + Cy$
- ③ ADC A, (HL) ... $A \leftarrow A + (HL) + Cy$
- ④ SBC n ... $A \leftarrow A - n - Cy$
- ⑤ SBC r ... $A \leftarrow A - r - Cy$
- ⑥ SBC (HL) ... $A \leftarrow A - (HL) - Cy$
- ⑦ ADC HL, rr ... $HL \leftarrow HL + rr + Cy$
- ⑧ SBC HL, rr ... $HL \leftarrow HL - rr - Cy$

n は 1 バイトのデータ、r は A、B、C、D、E、H、L レジスタを表します。rr は BC、DE、HL レジスタを表します。フラグの変化は、ADD 命令や SUB 命令と同じで、Cy フラグ、Z フラグのどちらも変化します。

Cy フラグは、繰り上がり、繰り下がりがあったときに 1 となり、それ以外は 0 となっていますから、Cy フラグの値は 1 か 0 ということになります。キャリー付き加算を縦書き計算してみます。

0011	0100	←	Aレジスタ
0010	0010	←	Bレジスタ
+		←	Cyフラグ
	1		
01101	0111	→	Aレジスタ

└──

→ 繰り上がり → Cyフラグ

さて、この命令を使うとどのようなことができるのでしょうか。
図3-7 をみてください。これは、2バイト長の数値の足し算です。
 ①はレジスタペアを使った足し算で、②はそれと同じ処理を、8ビットレジスタで行ったものです。②でやっていることは単純で、初めにCレジスタの内容とLレジスタの内容を足した後、今度はBレジスタの内容とHレジスタの内容を足しているだけです。しかし、もし初めの計算で繰り上がりがあった場合、次の計算でそれを計算に入れてやらなければなりません。そこで、ADC 命令を使っているのです（**図3-8**）。

このようにすれば、何バイトのデータの足し算でも可能です。最初に、最下位のバイトどうしを ADD 命令を使って計算しておき、残りは下位バイトより順に ADC 命令を使って計算していけばいいわけですからね（**リスト3-16**）。

HL ← HL+BC

① ADD HL, BC

② LD A, C

ADD A, L

LD L, A

LD A, B

ADC A, H

LD H, A

図3-7 2バイト長の数値の足し算

	ADC 命令		ADD 命令	
H →	0100 0101		1000 0000 ← L	
B →	1001 0000		1101 0011 ← C	
	+ 1		+ ← Cy	
	1101 1011		0101 0011	

図3-8 ADC命令の役割

リスト3-16 プログラム例

```

100 ' ; **** ADCメイレイ ****
110 '
120 '      ORG      0D0000H
130 '
140 '      LD       HL, 0FFFFH
150 '      LD       BC, 0003H
160 ' ; L=C+L
170 '      LD       A, C
180 '      ADD      A, L
190 '      LD       L, A
200 ' ; H=B+H+Cy
210 '      LD       A, B
220 '      ADC      A, H
230 '      LD       H, A
240 '
250 '      JP       0EB00H

```

SBC 命令でも行ってみましょう。SBC 命令では、Cy フラグは足すのではなく、引くことになります。これは、減算命令における Cy フラグが、繰り下がりを意味することによるものです。

それでは、ADD 命令のときと同じように、2 バイト長の引き算を行ってみましょう。レジスタペアに対する、2 バイト長の SUB 命令はないので、かわりとして使うこともできます。

処理としては、図 3-9 の(イ)で L レジスタの内容から C レジスタの内容が引かれます。この場合 0 から 1 は引けませんから Cy フラグを 1 にして、上の位から 1 ケタ借りてきたことにして計算されます。次に(ロ)で H レジスタの内容から B レジスタの内容が引かれますが、このとき Cy フラグが 1 になっていますから、これも一緒に引かれることになります(図 3-10)。

何バイトも使っているデータの引き算では、最初に最下位のケタどうしを SUB 命令で計算して、それから SBC 命令で残りのケタを計算します (リスト 3-17)。

HL ← HL-BC

- ①レジスタペアの SUB 命令はない
- ②LD A, C
 SUB L … (イ)
 LD L, A
 LD A, B
 SBC A, H … (ロ)
 LD H, A

図3-9 2バイト長の数値の引き算

	SBC 命令		SUB 命令	
H →	1111 1111		0000 0000	← L
B →	0000 0000		1000 0000	← C
	— 1		—	← Cy
	1111 1110		1000 0000	

図3-10 SBC命令の役割

リスト3-17 プログラム例

```
100 ;**** SBCメイレイ ****
110 ;
120 ;      ORG      0D000H
130 ;
140 ;      LD       HL, 0000H
150 ;      LD       BC, 0003H
160 ; ; L=C-L
170 ;      LD       A, C
180 ;      SUB      L
190 ;      LD       L, A
200 ; ; H=B-H-Cy
210 ;      LD       A, B
220 ;      SBC      A, H
230 ;      LD       H, A
240 ;
250 ;      JP       0EB00H
```


●ちょっと特殊な加算、減算命令

加算命令、減算命令の特殊なものとして INC (インクリメント) 命令と DEC (デクリメント) 命令があります。これはレジスタに対し、1 を足したり、引いたりする命令です。

① r レジスタに 1 を足す

INC r ... $r \leftarrow r + 1$

② HL レジスタの内容の示すアドレスの内容に 1 を足す

INC (HL) ... (HL) \leftarrow (HL) + 1

③ r レジスタから 1 を引く

DEC r ... $r \leftarrow r - 1$

④ HL レジスタの内容の示すアドレスの内容から 1 を引く

DEC (HL) ... (HL) \leftarrow (HL) - 1

⑤ rr レジスタに 1 を足す

INC rr ... $rr \leftarrow rr + 1$

⑥ rr レジスタから 1 を引く

DEC rr ... $rr \leftarrow rr - 1$

r は A、B、C、D、E、H、L レジスタを表します。rr は BC、DE、HL レジスタを表します。(HL) は HL レジスタの指すメモリの内容を表します。

これらの命令は、ADD 命令、SUB 命令などを使えば必要なさそうですが、実行速度が速くなる、ほとんどのレジスタが使えるなどの利点があります。

INC 命令、DEC 命令は、他の加算、減算命令とはフラグの変化が異なり、Cy フラグがまったく変化しません。Z フラグは 1 バイトのレジスタの場合変化しますが、レジスタペアでは変化しません。

INC 命令、DEC 命令のプログラム例を示します(リスト 3-18)。

リスト3-18 プログラム例

```
100 ' ;**** INC,DECメイレイ ****
110 '
120 '      ORG      0D0000H
130 '
140 '      LD       A,0
150 '      INC      A
160 '
170 '      LD       B,1
180 '      DEC      B
190 '
200 '      LD       HL,1000H
210 '      INC      HL
220 '
230 '      LD       DE,1001H
240 '      DEC      DE
250 '
260 '      JP       0EB00H
```

実行すると、Aレジスタが+1 され、Bレジスタが-1 されます。このとき Bレジスタは 0 になるので、Z フラグがセットされます。次に HL レジスタが+1 され、DE レジスタが-1 されます。Cy フラグは、これを実行する前の状態が持続されます。

プログラムの 流れを 変えてみよう



4.1 分岐してみよう

分岐とは、プログラム処理の流れを変えることをいい、BASICでは「GOTO文」、「GOSUB～RETURN文」がこれにあたります。分岐命令を覚えることにより、プログラムを繰り返し実行したり、サブルーチンを作ったりすることができるようになります。次に、それに相当するマシン語の命令を示します。

① 指定したプログラムへ分岐する

・ BASIC 言語

書式 GOTO <行番号>

例) GOTO 1000

・ アセンブリ言語

書式 JP <アドレス>

例) JP 0D000H

② 指定したサブルーチンをコールする。

・ BASIC 言語

書式 GOSUB <行番号> ... RETURN

例) GOSUB 2000 ... RETURN

・ アセンブリ言語

書式 CALL <アドレス> ... RET

例) CALL 0D000H ... RET

マシン語はメモリに置かれ実行されますが、メモリには行番号などというものはありません。ですから、そのかわりに分岐先をアドレスで示すことになります。

なお、アセンブリ言語のプログラムリストにある行番号は、アセンブラにとってはまったく意味のないものですから、次のようにしても30行に分岐することにはなりません。まちがえないでください。

×	10	'	JP	30
	20	'	LD	A, 0
	30	'	LD	B, 0

● JP 命令

JP(ジャンプ)命令は、まさに GOTO 文に相当する命令で、実行すると指定したアドレスに分岐して、そこから再びプログラムの実行を始めます。

① nn 番地にジャンプする

JP nn ... PC ← nn

PCレジスタに nn を代入する。フラグは変化しない。

② HL レジスタの内容の番地にジャンプする

JP (HL) ... PC ← HL

PCレジスタに、HL レジスタの内容を代入する。フラグは変化しない。

PC(プログラムカウンタ)というレジスタには、2 バイトのアドレスを表す数値が入っています。PC レジスタは、いつでも次に実行される命令のアドレスを指していて、1 命令実行されるごとに、自動的にその命令の必要とするバイト数が加算され、更新されていくようになっています。逆に言うならば、コンピュータは PC レジスタが指しているアドレスの内容を命令だとして、実行しているに過ぎません。ですから、PC レジスタを書きかえてしまえば、まったく別のアドレスにあるプログラムを、次に実行することができるわけです。JP 命令とは、この PC レジスタを書きかえる命令です。

LD 命令で“(HL)”と書いた場合は、HL レジスタが示すアドレスの内容でしたが、JP 命令で“(HL)”とした場合は HL レジスタの内容そのものを指します。

● CALL 命令と RET 命令

CALL (コール) 命令は、BASIC の GOSUB 文に相当する命令で、実行すると、指定されたサブルーチンに分岐します。その後、RET (リターン) 命令に出会うと、CALL 命令の次の命令に戻って、そこから実行し始めます。

① サブルーチンを呼び出す

```
CALL nn    ...    (SP-1) ← PC の上位バイト  
              (SP-2) ← PC の下位バイト  
              SP ← SP-2  
              PC ← nn
```

PCレジスタを対象に PUSH 命令と同じことをした後、nn 番地に分岐する。フラグは変化しない。

② サブルーチンから戻る

```
RET          ...    PC の上位バイト ← (SP)  
              PC の下位バイト ← (SP+1)  
              SP ← SP+2
```

PCレジスタを対象に POP 命令と同じことをする。フラグは変化しない。

BASIC では、RETURN 文があると、自動的に GOSUB 文の次に戻ってきます。しかし、それがどのような仕掛けによってなるのかまったくわかりません。ところが、マシン語では RET 命令で CALL 命令の次に戻れる理由が、明確になっています。

CALL 命令は、JP 命令と同じように、指定したアドレスに分岐します。しかし、ただ分岐するわけではなく、ジャンプする前に CALL 命令の次の命令のアドレス（つまり、PCレジスタの内容）を、スタックに記憶しておきます(PUSH 命令と同じことを自動的に行う)。やがて RET 命令に出会うと、ここでは逆にスタックから 2 バイト読み出し(POP 命令と同じことを自動的に行う)その値を戻りアドレスだとして、分岐するようになっていきます。

スタックは PUSH 命令、POP 命令で使われる他に、CALL 命令の戻りアドレスを記憶しておくという役目も持っています (図 4-1)。

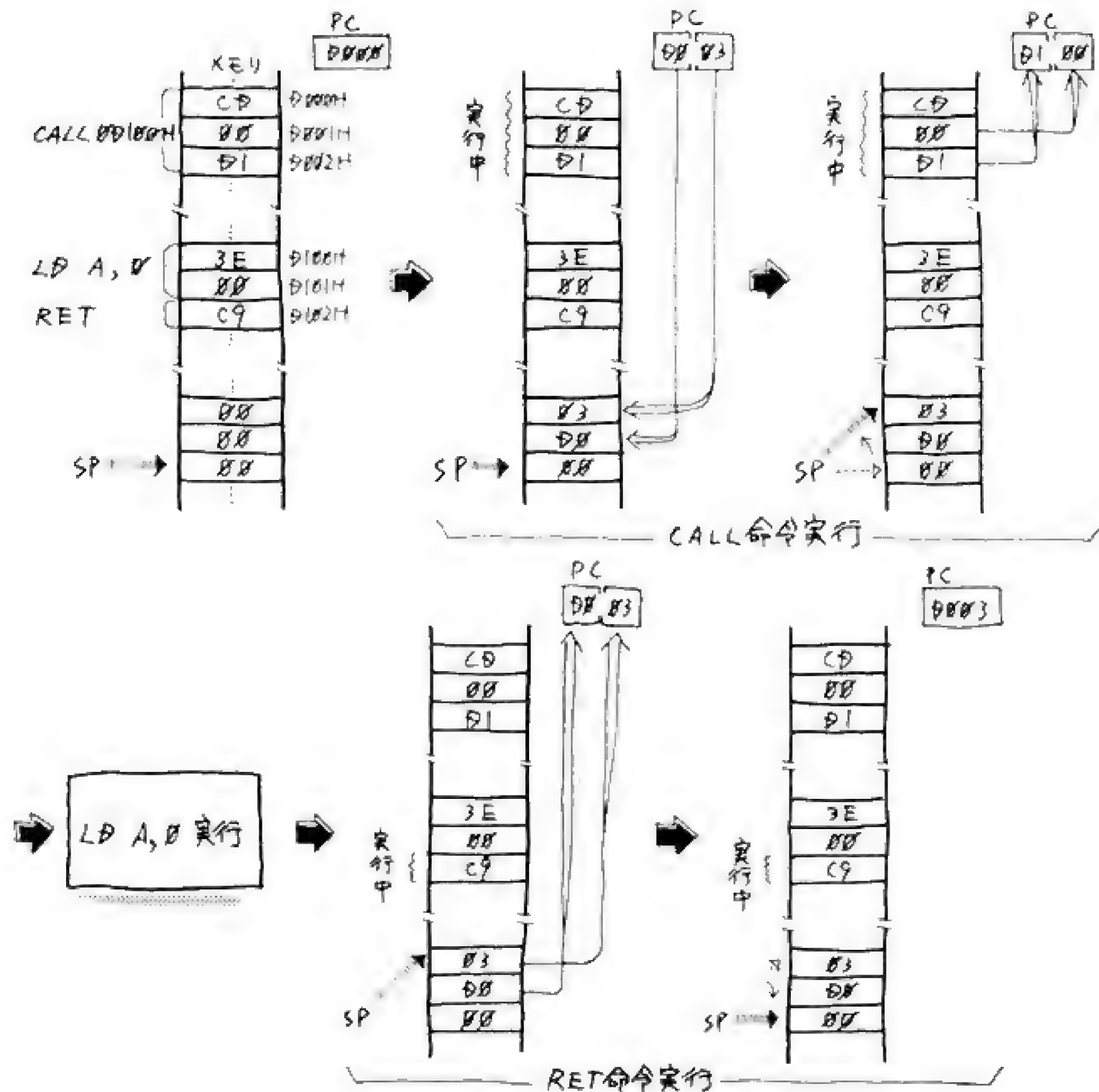


図4-1 スタックの使い方

● スタックが破壊される (その2)

P 64 でスタックが破壊される例を話しました。しかし、あのようなまちがいは、気を付ければすぐに見つけることができます。また、運がよければ何とか動作することもあります。しかし、CALL 命令、RET 命令の場合は、てき面にとんでもない結果が現れます。スタックが破壊されているにもかかわらず、RET 命令を実行してしまったら、何だかわからないデータを戻りアドレスだとして分岐してしまうので、簡単に暴走してしまうからです。

スタックは自分で管理するものです。慎重に慎重を重ねて使うようにしましょう。

● ラベルを使おう（その2）

アセンブリ言語でプログラムを作っているときに、これらの分岐命令を使おうとすると、大変困ってしまうことがあります。それは、分岐先のアドレスが、プログラムをひとめ見ただけではわからないということです。つまり、アセンブルして実際マシン語にしてみないことには、どの命令が何番地に置かれるのか不明確なので、分岐先を何番地にしたらいいのか、わからないということです。

```
<例 1>   ORG  OD000H
           JP   XXXXH      . . . . ①
           LD   A, 0
           LD   B, 0      . . . . ②
```

①から②へ分岐させたいけど、②の命令が何番地になるのかわからない。困ったなあ。

こんなとき、ハンドアセンブルでマシン語に変換する場合は、例2のようにすることになります。

```
<例 2>                                     ORG  OD000H
D000H  C3  XX  XX      JP   XXXXH
D003H  3E  00          LD   A, 0
D005H  06  00          LD   B, 0
```

JP 命令は3バイト命令で、LD 命令は2バイト命令だから、JP 先はD005H番地か。

このような方法でプログラムを作っていくのは、なかなか大変です。そこでアセンブラでは、「ラベル」を使うことで、分岐先を何番地にするか意識することなくプログラミングすることができるようになっています。

ここでのラベルは、基本的に次のように使われます。

```

        JP    <分岐先のラベル>
        :
        :
<ラベル>:    <分岐先の命令>

```

オペランドに書かれたラベルは、アセンブルしたとき、自動的にアドレスと置きかえられます。ラベルを使った例を示します（リスト4-1）。

リスト4-1 プログラム例

```

100 ' ;**** JPメイレイ 1 ****
110 '
120 '      ORG      0D000H
130 '
140 '      JP       USAGI ;1
150 ' LACCO:JP      KAME  ;3
160 ' USAGI:JP      LACCO ;2
170 ' KAME: LD      B,0   ;4
180 '
190 '      JP       0EB00H

```

このプログラムの実行すると、“JP USAGI”、“JP LACCO”、“JP KAME”の順に実行されます。

この場合のラベルは、EQU 命令などで定義する必要はありません。しかし、そのかわりにラベルの後ろには、必ず “: (コロン)” を付けないと、アセンブルエラーになってしまいます。その他は P 65 の書き方と同じです。

● 分岐命令の演習

では、実際に JP 命令とラベルを使ってプログラムを作ってみましょう（リスト4-2）。

リスト4-2 プログラム例

```

100 ' ;**** JPメイレイ 2 ****
110 '
120 '      ORG      0D000H
130 '
140 '      LD       BC,0
150 '      JP       TOBU .....①
160 '
170 '      LD       B,0BBH
180 ' TOBU: LD      C,0CCH .....②
190 '
200 '      JP       0EB00H

```

このプログラムでは、まず BC レジスタを 0 に初期化します。もし、次に JP 命令がなければ、B レジスタ、C レジスタにはそれぞれ 0BBH、0CCH が代入されるはずですが、JP 命令で①から②に分岐してしまうので、実際には B レジスタへの代入は行われません。

次に CALL 命令と RET 命令のプログラムを、ラベルを使って作ってみましょう（リスト 4-3）。

リスト4-3 プログラム例

```

100 ' ;**** CALL,RETメイレイ ****
110 '
120 ' ;メインルーチン
130 '      ORG      0D000H
140 '
150 '      LD       A,03H
160 '      CALL    BAI
170 '      LD       (0D100H),A
180 '      JP       0EB00H
190 '
200 ' ;サブ ルーチン
210 '      ORG      0D080H
220 '
230 ' BAI:  ADD     A,A
240 '      RET

```

D080H 番地からのプログラムは、Aレジスタの内容を2倍するサブルーチンです。AレジスタとAレジスタの内容を足し、結果がAレジスタに入るので、Aレジスタの内容が2倍になったのと同じことになります。

D000H 番地からのメインルーチンで、AレジスタにD100H番地のメモリの内容を代入し、サブルーチンを呼び出した後に、再びD100H番地にAレジスタの内容を代入しているため、結果的にD100H番地の内容が2倍されることになります。

● JR 命令

JP 命令の特殊なものとして、この JR (ジャンプ・リラティブ) 命令があげられます。JR 命令はアセンブリ言語レベルで、JP 命令とまったく同じように使うことができます。では、JP 命令と JR 命令を詳しく見るために、両命令をアセンブルしてみましょう (リスト 4-4)。

リスト4-4 アセンブルリスト(プリンタ出力)

100:			;**** JRメイレイ 1 ****
110:			
120:	D000	ORG	0D000H
130:			
140:	D000	1803	JR TONDA
150:	D002	C305D0	JP TONDA
170:	D005	TONDA:	
190:	D005	C300EB	JP 0EB00H
	D005	TONDA	

JR 命令のオペランドが、1 バイトであることに注意してください。JP 命令を使うより JR 命令を使った方が、プログラムが小さくなりますね。

これは、JP 命令が「絶対ジャンプ」しているのに対し、JR 命令は「相対ジャンプ」している結果です。

絶対ジャンプでは、分岐先をアドレスで直接指定しましたが、それに対し相対ジャンプでは、その命令からの距離で指定します。

① 自分のアドレスから相対的に分岐する

・マシン語での処理

JR e ... $PC \leftarrow PC + e$

この JR 命令の次の命令のアドレスに、e を加えたアドレスへジャンプします。e は $-128 \sim 127$ の範囲の数値を表します。フラグは変化しません。なお、マシン語ではオペランドを相対値で指定しますが、アセンブリ言語では JP 命令と同じように絶対アドレスでプログラミングします。

いままで 00H~FFH までを、10 進数の 0~255 に対応させて考えていましたが、相対ジャンプの場合は 00H~7FH を 0~127 に、80H~FFH を $-128 \sim -1$ に対応させて考える必要があります。これを「2 の補数表現」といいます (図 4-2)。

こうすることによって、アドレスの大きい方に 127 バイト、アドレスの小さい方に 128 バイト分岐できるわけです (図 4-3)。

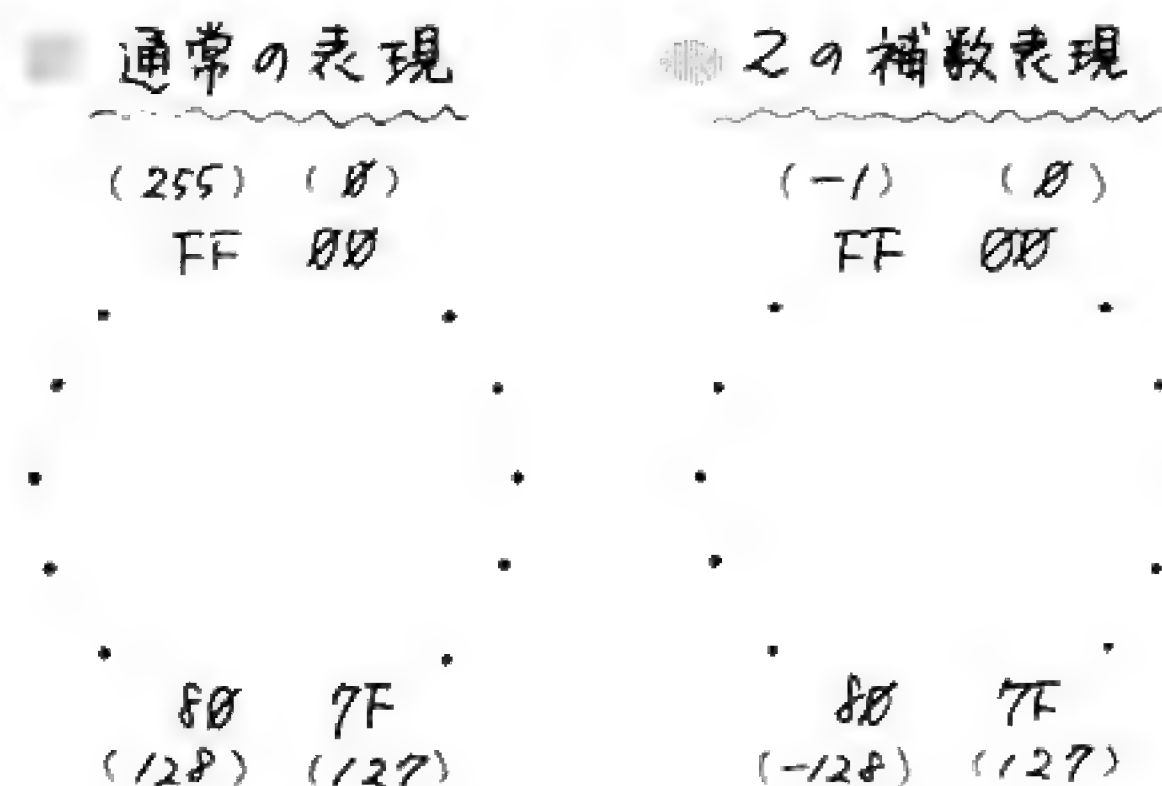


図4-2 2の補数表現

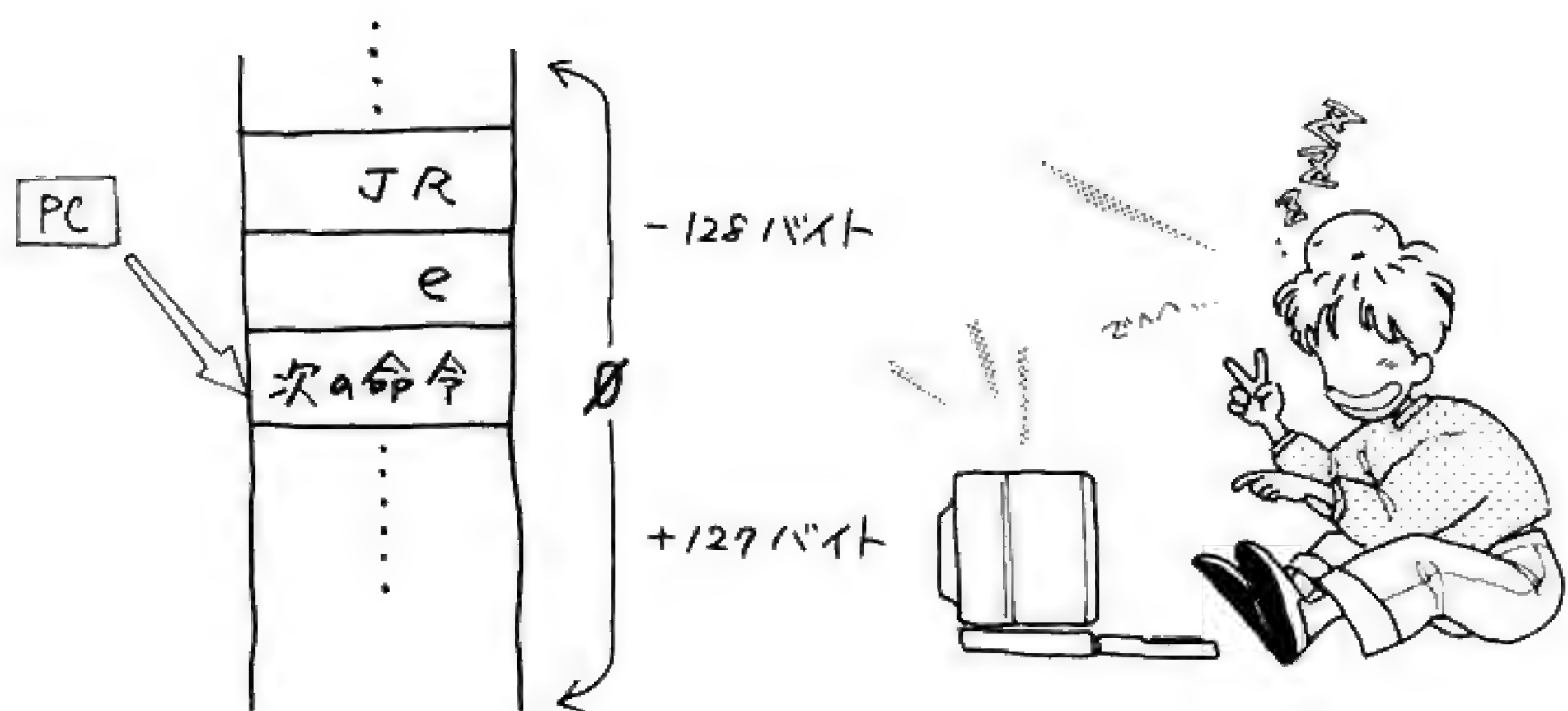


図4-3 JR命令でジャンプできる範囲

アセンブリ言語で JR 命令を使うときは、JP 命令と同じように絶対アドレスで書くことになっています。このようにしておくと、アセンブラが分岐先と次の命令のアドレスとの差を計算して、マシン語にしてくれます。ただし、そのときアドレスが JR 命令で分岐できないようであれば、アセンブルエラーとなります(リスト 4-5)。

リスト4-5 アセンブルエラーリスト(プリンタ出力)

100:		;**** JRメイレイ(エラー) ****	
110:			
120:	D000	ORG	0D000H
130:			
140:	Reference Error		
140:	R D000 18FE	JR	TOBU
150:			
160:	D100	ORG	0D100H
170:			
180:	D100 0600	TOBU:LD	B,0
190:	D102 C300EB	JP	0EB00H

JR 命令は完全に JP 命令で置き換えることができるので、初心者の方は、エラーを出さないためにも、JR 命令を使うのは、ひかえた方がいいでしょう。

マシン語には、このような相対的な指定ができる命令が、少しあります。しかし、それらを使わなくても、プログラミングに支障な

いので、JR 命令以外はあえて説明しません。このような考え方があるという程度に覚えておいてください。

4.2 判定してみよう

ここでは、BASIC の IF～THEN 文に相当するプログラムを、マシン語で作ってみましょう。マシン語での判定は、いままで出てきたものよりも、少し難しいかもしれません。急がずにユックリ見ていくことにします。

① BASIC での判定

書式 IF <条件> THEN GOTO <行番号>

例) IF A=0 THEN GOTO 1000

② アセンブラでの判定

書式 JP <条件>, <アドレス>

例) JP Z, 0D000H

BASIC では、<条件>として、次のような条件式を書くことができ、もしその条件を満たしていたならば<行番号>へ分岐します。

A = B	左辺と右辺が等しいなら
A < B	左辺が右辺より小さいなら
A > B	左辺が右辺より大きいなら
A <= B	左辺が右辺と等しいか、右辺より小さいなら
A >= B	左辺が右辺と等しいか、右辺より大きいなら
A <> B	左辺と右辺が等しくないなら

マシン語では、分岐命令で出てきた JP 命令を使います。この命令は、<条件>を指定することで、条件付きの分岐命令になります。そして、その条件が満たされた場合だけ<アドレス>に分岐します。条件が満たされない場合は、JP 命令の次の命令を実行します。

マシン語で指定できる<条件>には、次のようなものがあります。

Z (ゼロ)	Z フラグがセット (1) なら
NZ (ノンゼロ)	Z フラグがリセット (0) なら
C (キャリー)	Cy フラグがセット (1) なら
NC (ノンキャリー)	Cy フラグがリセット (0) なら

(他のフラグでも同じような条件が指定できますが、この本では省略します)

このように、マシン語ではフラグの状態を判定することによって分岐しますから、ある数値と、ある数値を比べるときも結果をフラグの変化として現れるようにする必要があります。

● 条件付き分岐命令

この命令は一般に以下のように書かれます

① 条件付き JP (ジャンプ) 命令

JP CC, nn ... CC が成り立てば $PC \leftarrow nn$

② 条件付き JR (ジャンプ・リラティブ) 命令

JR CC, e ... CC が成り立てば $PC \leftarrow PC + e$

アセンブリ言語で JR 命令を使うときは、前に説明したように JP 命令とまったく同じに書きます。

CC (コンディションコード) は条件 (Z、NZ、C、NC) を表し、もし CC で表される条件フラグと実際のフラグの状態が同じであれば分岐することになります。JP 命令を例にとつて、条件を実際書いてみます。

① ジャンプキャリー

JP C, XXXXH

Cy フラグがセットなら XXXXH に分岐、そうでなければ次の命令へ行く。

② ジャンプノンキャリー

JP NC, XXXXH

Cy フラグがリセットなら XXXXH に分岐、そうでなければ次の命令へ行く。

③ ジャンプゼロ

JP Z, XXXXH

Z フラグがセットなら XXXXH に分岐、そうでなければ次の命令へ行く。

④ ジャンプノンゼロ

JP NZ, XXXXH

Z フラグがリセットなら XXXXH に分岐、そうでなければ次の命令へ行く。

判定を行う上で大事なことは、いかにしてフラグを変化させるかということです。それについては P97 から行います。

● 条件付き CALL 命令、RET 命令

JP 命令と同じように、CALL 命令、RET 命令でも<条件>を指定することができます。条件付きの CALL 命令、RET 命令は、BASIC でいうと次のような文に相当します。

① 条件付き CALL 命令

・ BASIC 言語

書式 IF <条件> THEN GOSUB <行番号>

例) IF A=B THEN GOSUB 1000

・ アセンブリ言語

書式 CALL <条件>, <アドレス>

例) CALL C, 0D100H

② 条件付き RET 命令

・ BASIC 言語

書式 IF <条件> THEN RETURN

例) IF A=B THEN RETURN

・ アセンブリ言語

書式 RET <条件>

例) RET C

条件付きのCALL 命令とRET 命令では、実際には以下のようなことが行われます。CC は、JP 命令で示した条件と同じものです。

① 条件付き CALL 命令

CALL	CC, nn	...	CC が成り立てば
			(SP-1) ← PC の上位バイト
			(SP-2) ← PC の下位バイト
			SP ← SP-2
			PC ← nn

条件付き RET 命令

RET	CC	...	CC が成り立てば
			PC の上位バイト ← (SP)
			PC の下位バイト ← (SP+1)
			SP ← SP+2

● IF~THEN 文のように (その 1)

条件付きの分岐命令では、フラグの状態だけしか条件として指定できません。BASIC には、もともとフラグという考えがないので、たとえば A と B が等しいかどうかを、どう判定させたらいいのか、このままではわからないと思います。

そこで、ここでは条件付き分岐命令の応用例として、BASIC のいろいろな条件文に対応するプログラムを、アセンブリ言語で作ってみましょう。

次の IF～THEN 文を見てください。

```
IF A=B THEN GOTO XXXX
```

この場合、もし A と B が等しかったならば、A から B を引くと 0 になるはずです。この性質を利用して、プログラムを作ってみると、次のようになります。

```
SUB B
JP Z,XXXXH
```

SUB 命令で、Aレジスタから Bレジスタを引きます。もし Aレジスタと Bレジスタの内容が同じならば結果は 0 になり、Z フラグがセットされます。JP 命令では、その Z フラグの状態を見て、セットされていれば XXXXH へ分岐し、そうでなければ次の命令に行きます。

条件付き分岐のプログラム例を次に示します（リスト 4-6）。

リスト4-6 プログラム例

```
100 ' ;**** ハンテイ 1 ****
110 '
120 '      ORG      0D000H
130 '
140 '      LD       A,3
150 '      LD       B,3
160 '
170 '      SUB      B
180 '      JP       Z,MOMO
190 '
200 '      LD       B,0FFH
210 '
220 ' MOMO: JP      0EB00H
```

まず、Bレジスタの値を判定します。もし Aレジスタと等しければ分岐し、何もせずに終了します。そうでなければ Bレジスタに FFH が設定されて終了します（この場合は、Aレジスタと Bレジスタが等しいので、必ず分岐します）。

条件付きの CALL、RET 命令についても行ってみましょう(リスト 4-7)。

リスト4-7 プログラム例

```

100 ' ; **** ハンディ 2 ****
110 '
120 '      ORG      0D0000H
130 ' ; メインルーチン
140 '      LD       A, 0
150 '      LD       C, 0
160 '
170 '      SUB      C
180 '      CALL     Z, MIKAN
190 '
200 '      JP       0EB00H
210 '
220 ' ; サブ ルーチン
230 ' MIKAN:
240 '      LD       C, 0FFH
250 '      RET

```

Cレジスタの値が 0 ならば、サブルーチンをコールします。そうでなければ、何もされずに終了します。

マシン語で IF~THEN 文を実現させるには、このように条件に応じてフラグを変化させる部分と、そのフラグの状態をみて条件分岐させる部分に分けて書きます。

フラグを変化させる命令には、SUB 命令の他にもいくつかあります。それらを以下にあげておきます。詳しくは P140 のマシン語命令表を見てください。

実行することにより Z、Cy フラグが変化する命令(この本に出ている命令のみ)をあげてみます。

① 8ビットレジスタが対象の場合

・ Z、Cy フラグが変化する命令

ADD 命令、ADC 命令、SUB 命令、SBC 命令、CP 命令

・ Z フラグだけ変化する命令

INC 命令、DEC 命令

② 16ビットレジスタが対象の場合

- ・ Z、Cy フラグが変化する命令
ADC 命令、SBC 命令
- ・ Cy フラグだけ変化する命令
ADD 命令

● CP 命令

P ●●の例を見ると、BASIC 側は変数 A の値が変化しないのに対し、アセンブリ言語のプログラムでは、A レジスタが変化してしまいます。もちろん、あらかじめ A レジスタの内容を、別のレジスタかメモリに記憶させた後で戻せば問題ありませんが、判定するたびにそれでは、いくら何でもめんどうです。

マシン語には、そんなときのために CP (コンペア) 命令という命令が用意されています。

① A レジスタと n を比較する

CP n ... A-n

② A レジスタと r レジスタの内容を比較する

CP r ... A-r

r は、A、B、C、D、E、H、L レジスタを表します。

CP 命令は、一般に「比較命令」と呼ばれ、フラグだけ変化させたい場合に使います。働きは、SUB 命令とほとんど同じですが、計算結果が A レジスタに入らないので、A レジスタの内容は変化しません。要するに、計算結果はなくなってしまい、フラグだけが変化します。フラグは Cy、Z のいずれも変化します。

前の例を CP 命令を使って書き直してみましょう(リスト 4-8)。

```
CP  B
JP  Z,XXXXH
```


リスト4-8 プログラム例

```

100 ' ;**** CPメイレイ ****
110 '
120 '      ORG      0D000H
130 '
140 '      LD       A,3
150 '      LD       B,3
160 '
170 '      CP       B
180 '      JP       Z,MOMO
190 '
200 '      LD       B,0FFH
210 '
220 ' MOMO: JP      0EB00H

```

Aレジスタの内容が変化しないことに注意してください。

● IF～THEN 文のように（その2）

比較、判定のまとめとして、IF～THEN 文に相当するマシン語プログラムを<条件>ごとに書き出してみます。

① A と B が等しいとき処理 1 へ

```

10  IF A=B THEN GOTO <処理 1 へ>
20  <処理 2>
      ↓
      CP B
      JP Z,XXXXH      ; 処理 1 へ分岐
      <処理 2>

```

AレジスタからBレジスタを引いてみて、結果が0であればXXXXHへ分岐する。

② A と B が等しくないとき処理 1 へ

```

10  IF A<>B THEN GOTO <処理 1 へ>
20  <処理 2>
      ↓

```



```
CP B
JP NZ,XXXXH ;処理1へ分岐
<処理2>
```

AレジスタからBレジスタを引いてみて、結果が0でなければXXXXHへ分岐する。

③ BよりAが小さいとき処理1へ

```
10 IF A<B THEN GOTO <処理1へ>
20 <処理2>
    ↓
    CP B
    JP C,XXXXH ;処理1へ分岐
    <処理2>
```

AレジスタからBレジスタを引いてみて、結果が0よりも小さくなったら（繰り下がりがあったら）XXXXHへ分岐する。

④ BとAが等しいか、BよりAが大きいとき処理1へ

```
10 IF A>=B THEN GOTO <処理1へ>
20 <処理2>
    ↓
    CP B
    JP NC,XXXXH ;処理1へ分岐
    <処理2>
```

AレジスタからBレジスタを引いてみて、結果が0か、0よりも大きくなったら（繰り下がりがなかったら）XXXXHへ分岐する。

⑤ BよりAが大きいとき処理1へ

```
10 IF A>B THEN GOTO <処理1へ>
20 <処理2>
    ↓
```



```

CP B
JP Z, NEXT      ; 処理 2 へ分岐
JP NC, XXXXH    ; 処理 1 へ分岐

```

NEXT: <処理 2>

AレジスタからBレジスタを引いてみて、結果が0でなく、しかも0より大きくなったら(繰り下がりがなかったら)XXXXHへ分岐する。

⑥ BとAが等しいか、BよりAが小さいとき処理 1 へ

```

10 IF A<=B THEN GOTO <処理 1 へ>

```

```

20 <処理 2>

```

↓

```

CP B
JP C, XXXXH      ; 処理 1 へ分岐
JP Z, XXXXH      ; 処理 1 へ分岐

```

<処理 2>

AレジスタからBレジスタを引いてみて、結果が0よりも小さいか(繰り下がりがあるか)、0であつたらXXXXHへ分岐する。

⑦ Aが0のとき処理 1 へ

```

10 IF A=0 THEN GOTO <処理 1 へ>

```

```

20 <処理 2>

```

↓

```

CP 0
JP Z, XXXXH      ; 処理 1 へ分岐

```

<処理 2>

Aレジスタから0を引いてみて、結果が0であつたらXXXXHへ分岐する。

⑧ A が 0 でないとき処理 1 へ

10 IF A<>0 THEN GOTO <処理 1 へ>

20 <処理 2>

↓

CP 0

JP NZ, XXXXH ; 処理 1 へ分岐

<処理 2>

Aレジスタから 0 を引いてみて、結果が 0 でなかったら XXXXH へ分岐する。

さて、これらを見ていると、“>” と “<=” という条件だけは、そのままマシン語にすると、意外に大変だということがわかります。このような条件の場合は次のように直してしまった方が、プログラムが簡単になります。

A>3 → A>=4

A<=3 → A<4

なお、これらは CALL 命令や RET 命令でも、まったく同じように行うことができます。

● 16ビットデータを比較するには

8ビットデータの場合は、CP 命令という比較専門の命令がありますが、16ビットデータの場合はそのような便利な命令はありません。そこで、例として HL レジスタにあるデータを比較する方法を考えてみましょう。この方法には以下の 2 つがあります。

① 16ビットまとめて判定する

LD BC, 1234H … 比較対象データ

CP A … Cy を 0 にするため

SBC HL, BC … HL ← HL - BC

JP Z, XXXXH

16ビットレジスタに対する命令のうちZフラグが変化するものはADC命令とSBC命令だけです。比較を行うには、SBC命令を使います。ここでは、BCレジスタに比較対象のデータを入れておき、HLレジスタからそれを引くことで判定してみます。

“CP A”は、Cyフラグをリセットするためにあります。理由は、SBC命令実行のとき、Cyフラグをいっしょに引いてしまわないようにするためです。AレジスタからAレジスタを引くので、結果は必ず0になります。そこでZフラグはセットされますが、Cyフラグはリセットされます。

② 8ビットずつ分けて判定する

```
LD    A, 12H
CP    H
JP    NZ, NEXT
LD    A, 34H
CP    L
JP    Z, XXXXH
```

NEXT:

初めにHレジスタを判定し、もし12HであればLレジスタを判定します。この方法は、①に比べると少しめんどろですが、3バイト以上のデータに対しても応用がきくので、知っておいた方がいいでしょう。

では、①の方法を例にとって、プログラムを作ってみます（リスト4-9）。

リスト4-9 プログラム例

```
100 ' ;**** ハンテイ(16bit) ****
110 '
120 '      ORG      0D000H
130 '
140 '      LD       HL, 0E000H
150 '      LD       BC, 0E000H
160 '      LD       A, 0
```

```

170 '
180 '      CP      A
190 '      SBC    HL, BC
200 '      JP      Z, USAGI
210 '
220 '      LD      A, 0FFH
230 ' USAGI:
240 '      JP      0EB00H

```

HLレジスタからBCレジスタを引き、もし0ならば(等しければ) "USAGI"というラベルに分岐します。そうでなければ、AレジスタにFFHが代入されます。

この例では、初めにHLレジスタとBCレジスタを同じにしてあるので、必ず分岐することになります。

● 一般の IF～THEN～ELSE 文

BASICのIF～THEN文のもっとも典型的な形は、次の2種類です。

① IF <条件> THEN <文>

② IF <条件> THEN <文> ELSE <文>

これらを直接マシン語に直すのは難しいので、先ほどの"IF～THEN GOTO"という形に直し、最終的にアセンブリ言語にしてみましょう。

① 1000 IF A=B THEN <文>

1010 <次の文>

↓

1000 IF A<>B THEN GOTO 1010

1002 <文>

1010 <次の文>

↓

CP B


```

                JP    NZ, L1010
                <命令>
L1010:         <次の命令>

```

② 1000 IF A=B THEN <文 1> ELSE <文 2>
 1010 <次の文>

↓

```

1000 IF A<>B THEN GOTO 1006
1002 <文 1>
1004 GOTO 1010
1006 <文 2>
1010 <次の文>

```

↓

```

                CP    B
                JP    NZ, L1006
                <命令 1>
                JP    L1010
L1006:         <命令 2>
L1010:         <次の命令>

```

<文>は、実際には複数の文であってもかまいません。ラベル“L××××”は、わかりやすくするために行番号と同じにただけなので、他のラベルにしてもかまいません。

ここでは、もとの文の条件 (A=B) と、直した文の条件 (A<>B) がまったく正反対の条件になっていることに注意してください。なお、正反対の条件とは、次のような式をいいます。

正反対の条件

=	↔	<>
>	↔	<=
<	↔	>=

● 複雑な条件の IF～THEN 文

条件文の中がもっと複雑な場合を考えてみます。これも、より単純な BASIC に書きかえてからマシン語に直します。

① AND により 2 つ以上の条件を判定している IF 文の場合

```
1000 IF (A=B) AND (A=C) THEN <文>
```

```
1010 <次の文>
```

↓

```
1000 IF A<>B THEN GOTO 1010
```

```
1002 IF A<>C THEN GOTO 1010
```

```
1004 <文>
```

```
1010 <次の文>
```

↓

```
CP B
```

```
JP NZ, L1010
```

```
CP C
```

```
JP NZ, L1010
```

```
<命令>
```

```
L1010: <次の命令>
```

② OR により 2 つ以上の条件を判定している IF 文の場合

```
1000 IF (A=B) OR (A=C) THEN <文>
```

```
1010 <次の文>
```

↓

```
1000 IF A=B THEN GOTO 1004
```

```
1002 IF A<>C THEN GOTO 1010
```

```
1004 <文>
```

```
1010 <次の文>
```

↓

```
CP B
```



```

        JP      Z, L1004
        CP      C
        JP      NZ, L1010
L1004:  <命令>
L1010:  <次の命令>

```

4.3 ループしてみよう

ある条件が成立するまで、同じ作業を繰り返し行うことを「ループ」といいます。BASICでは、FOR～NEXT 文がこれにあたります。しかし、マシン語にはこのような、繰り返し専門の命令はありませんから「判定」する方法を応用して、そのかわりをさせることにします。

```

1000 FOR N=1 TO 10
1010 <文>
1020 NEXT N
1030 <次の文>

```

FOR～NEXT 文は、<文>を 1 回実行するごとに、変数 N に 1 を足し、その回数を数えます。そして、変数 N の内容が 10 になったところでループから抜けるようになっています。

これを IF～THEN 文に書き直すと次のようになります。

```

1000 N=1
1010 IF N=11 THEN GOTO 1050
1020 <文>
1030 N=N+1
1040 GOTO 1010
1050 <次の文>

```

(N=1、2、3・・・10 のときに<文>を実行)

第4章 プログラムの流れを変えてみよう

このときの変数 N のように、回数を数えるためのものを「カウンタ」といいます。この場合は、カウンタをどんどん足していった、ある数になったらループを抜けるようにしていますが、要するにループを何回繰り返したか数えるだけならば、次のように書いても同じです。

```
1000 N=10
1010 IF N=0 THEN GOTO 1050
1020 <文>
1030 N=N-1
1040 GOTO 1010
1050 <次の文>
(N=10、9、8・・・1 のときに<文>を実行)
```

カウンタに 10 を設定して、そこから実行するごとに 1 を引いていき、0 になったら終わりになります。このように書き変えた理由は、1010 行で変数 N が 11 であることを判定するよりも、0 であることを判定する方が、マシン語にしたときプログラムが簡単になるからです。

では、これに相当するマシン語のプログラムを作ってみます。まず、カウンタをメモリにするか、レジスタにするかを決めます。今回は、あつかいの簡単なレジスタにします。カウンタにはどのレジスタを使ってもかまいませんが、Aレジスタを選んだりとすると、8ビットの計算をしなければならないとき、わざわざカウンタの値を PUSH しなければなりません。

レジスタには、ある程度の役割というものがあります。はっきりと決まっているわけではありませんが、カウンタには BCレジスタ (255 回以内のループでは、そのどちらか) が使われることが多いようです。ここでは、Bレジスタにカウンタの役をつとめてもらいましょう。


```

                LD    B, 10
L1010:         JP    Z, L1050    ...①
                <命令>
                DEC   B
                JP    L1010
L1050:         <次の命令>

```

DEC 命令では、Bレジスタが0になったときに限り、Zフラグをセットします。JP 命令はフラグを変化させないので、フラグはDEC 命令のときの状態を保ちつつ①の JP 命令を実行し、Zフラグがセットされていれば、L1050 というラベルに分岐します（ただし、このループに入る前に Zフラグをリセットしておかないと、1 度もループせずに L1050 に分岐してしまうことがあります）。

このループをさらに簡単にしてみましょう。

```

                LD    B, 10
LOOP:         <命令>
                DEC   B
                JP    NZ, LOOP
                <次の命令>

```

もうおわかりですね。なお、当然のことながら、<命令>の中で B レジスタの値を変えてはいけません。

例として 1 から 10 まで、値を足すプログラムを作ってみます(リスト 4-10)。

リスト4-10 プログラム例

```

100 ' ;**** LOOP 1 ****
110 '
120 '      ORG      0D000H
130 '
140 '      LD       B, 10
150 '      LD       A, 0
160 '      LD       C, 1
170 '

```

```

180 ' LOOP: ADD      A, C
190 '               INC      C
200 '               DEC      B
210 '               JP       NZ, LOOP
220 '
230 '               JP       0EB00H

```

Bレジスタがカウンタです。Cレジスタの値はループするごとに1増えていき、そのたびにAレジスタにCレジスタの値が足されます。それを10回繰り返すことにより、Aレジスタには1から10まで足された数値が入ることになります。

● もう1つのループ

ループには、ある一定回数だけ繰り返すループの他に、ある条件が成立するまで繰り返すループがあります。このループには2つの方法があり、BASICで書くと次のようになります。

① 条件が成立していれば、<文>を1回も実行しなくていいときのループ（図4-4）

```

1000 IF A=B THEN GOTO 1030
1010 <文>
1020 GOTO 1000
1030 <次の文>

```

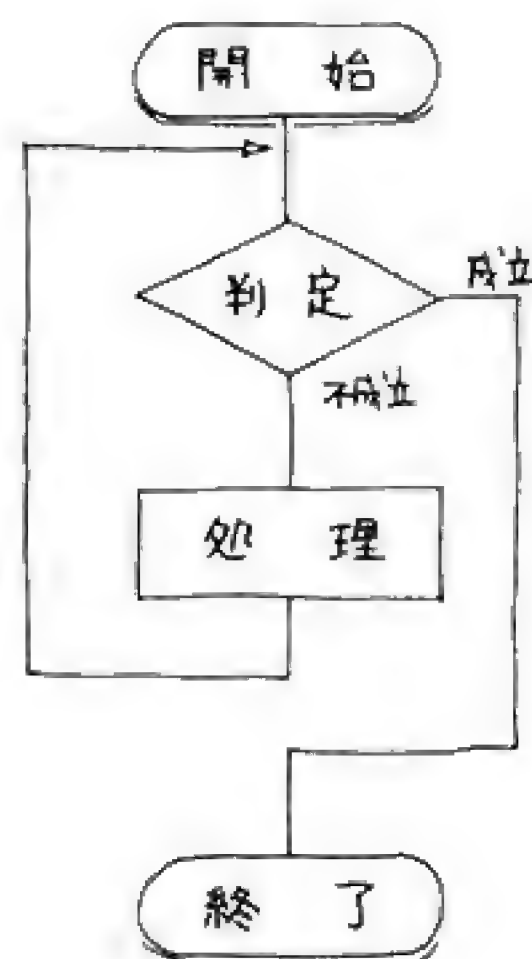


図4-4 フローチャート

② 条件が成立していても、必ず 1 回は<文>を実行したいときのループ (図 4-5)

```
1000 <文>
1010 IF A<>B THEN GOTO 1000
1020 <次の文>
```

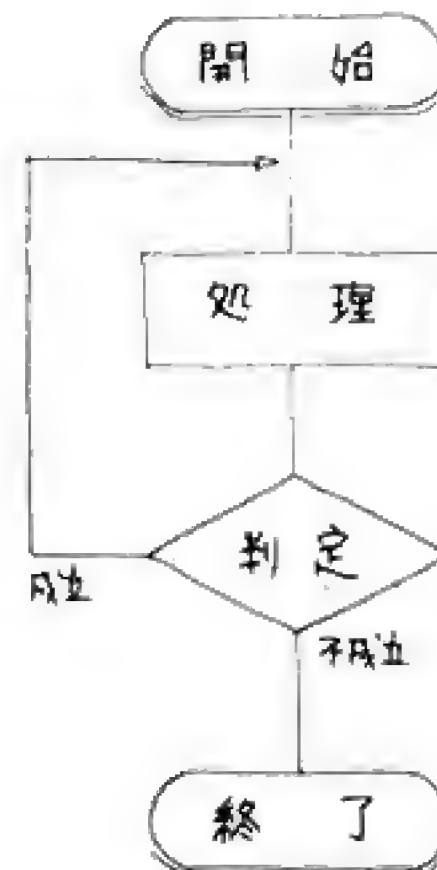


図4-5 フローチャート

これらに相当するマシン語のプログラムは、次のようになります。

```
① LOOP: CP      B
          JP      Z, EXIT
          <命令>
          JP      LOOP
EXIT:    <次の命令>
```

```
② LOOP: <命令>
          CP      B
          JP      NZ, LOOP
```

どちらも、A と B が等しければ、ループを抜けます。①では<命令>が実行される前に条件が判定されるため、1 回も<命令>を実行しないことがあります。それに対し②の場合は<命令>を必ず 1 回は実行します。

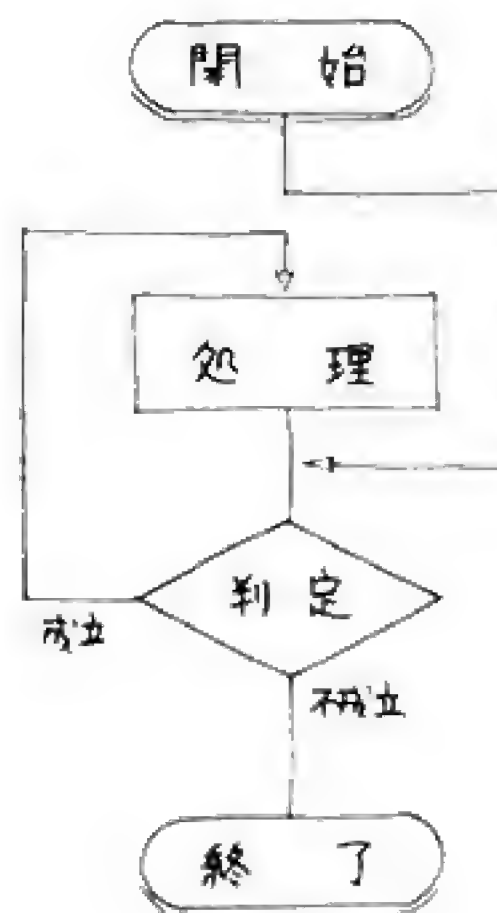
①は、次のような書き方もでき、こうすると②との条件の違いがよくわかります。

- ③ 条件が成立していても、かならず 1 回は<文>を実行したいときのループその 2 (図 4-6)

```

1000 GOTO 1020
1010 <文>
1020 IF A<>B THEN GOTO 1010
    
```

図4-6 フローチャート



```

                JP ENTER
LOOP:   <命令>
ENTER:  JP NZ, LOOP
    
```

①と②のプログラム例を示します。どちらも B レジスタの内容が 10 になるまでループを行うようになっています。よく見比べてみてください (リスト 4-11、リスト 4-12)。

リスト4-11 プログラム例

```

100 ' ;**** LOOP 2 ****
110 '
120 '      ORG      0D000H
130 '
140 '      LD       A,10H
150 '      LD       B,0FFH
160 '
170 ' LOOP: CP      B
180 '      JP       Z,EXIT
190 '      DEC      B
200 '      JP       LOOP
210 ' EXIT:
220 '      JP       0EB00H
    
```

リスト4-12 プログラム例

```

100 ' ;**** LOOP 3 ****
110 '
120 '      ORG      0D000H
130 '
140 '      LD       A, 10H
150 '      LD       B, 0FFH
160 '
170 ' LOOP: DEC     B
180 '      CP       B
190 '      JP       NZ, LOOP
200 ' EXIT:
210 '      JP       0EB00H

```

こらむ

暴走はどうしておこるのか

マシン語のプログラムを作る機会が多くなると、実行したとだんにリセットがかかったり、ウンともスンともいわなくなったりして、せつかくのプログラムが水の泡となることがあります。このような状態を「暴走」といいます（別に足がはえて走り回るわけではありませんよ）。

暴走した状態をマシン語からみると、例えば私たちが作ったプログラムが暴走してしまった場合は、次のどちらかの状態におちいつてしまっていると考えられます。

① プログラムが永久ループしてしまっている

永久ループとは、いつまでも同じ部分を実行して、ループから抜けないことをいいます。フラグの判定などを誤っているとおこしやすいので、判定するときはよく考えてから行いましょう。

② プログラム以外の場所に分岐してしまった

コンピュータは、PCレジスタの指すメモリの内容をマシン語の命令だとして、どんどん機械的に実行しているだけです。目的以外のアドレスに分岐してしまっても、実行が止まるわけではなく、わけのわからないデータをも命令だとして実行し続けます。暴走とは、本来このような状態をいいます。これは、プログラムやスタックを破壊するようなプログラムを実行した場合にもおこります。

なお、マシン語を実行したときに、エラーメッセージが出ることがありますが、これは暴走の際、たまたま BASIC のエラールーチンを実行してしまった結果です。このメッセージを信用してはいけません。

MSX は電源を入れると (リセットすると) 0000H 番地から実行し始めます。0000H 番地にはインタプリタへ分岐する命令が書かれているので、そのとき暴走することはありません。

入出力装置を コントロール してみよう



5.1 MSXの手足、入出力装置

今までに出てきた、レジスタやメモリといったものは、人にたとえるなら頭の中のできごとに過ぎません。ですから、いくらすごいことを考えようとも、それを表現する手段がなければ、その考えは何の役にもたたないことになります。また、逆に外からデータが入ってこなければ、状況の変化などを知り、それに応じて対処することもできません。そのため、人には口や鼻、耳、目、手足、といった、外とのやりとりをする部分がそなわっています。

MSXの場合もそこはまったく同じで、データを入力するために、キーボードやジョイスティックなどが、出力するために、モニタテレビやプリンタなどの周辺装置が使えるようになっています。ですから、これらをコントロールできなくては、マシン語がわかったとはいえないわけです。

では、それら入出力装置とやりとりを行う手始めとして、みなさんがもっとも興味のあると思われる、画面表示から始めてみましょう。

● 画面に文字を表示させたい

文字を表示することは、とても大切なことです。これができないと、必要なメッセージを表示したり、計算結果を知ることができません。いままで、メモリの中ばかり見てきたわけですから、ここらでパアッと自分の名前でも、マシン語を使って画面に表示させたい人もいることでしょう。

さっそく画面表示を行う、PRINT文に相当するような命令を紹介……といきたいところなのですが、困ったことにマシン語にそのような命令はないのです。さらに言ってしまうと、キーボードから入力した文字を読み取ったり、絵を書いたり、音をならしたりなど、特定の入出力装置とデータをやりとりするような命令はまったく存在しません。

では、どのようにしたら、このようなことができるのでしょうか。それには、MSXの中身と、その構成を知るのが一番です。図5-1にそれを示したので見てみましょう。

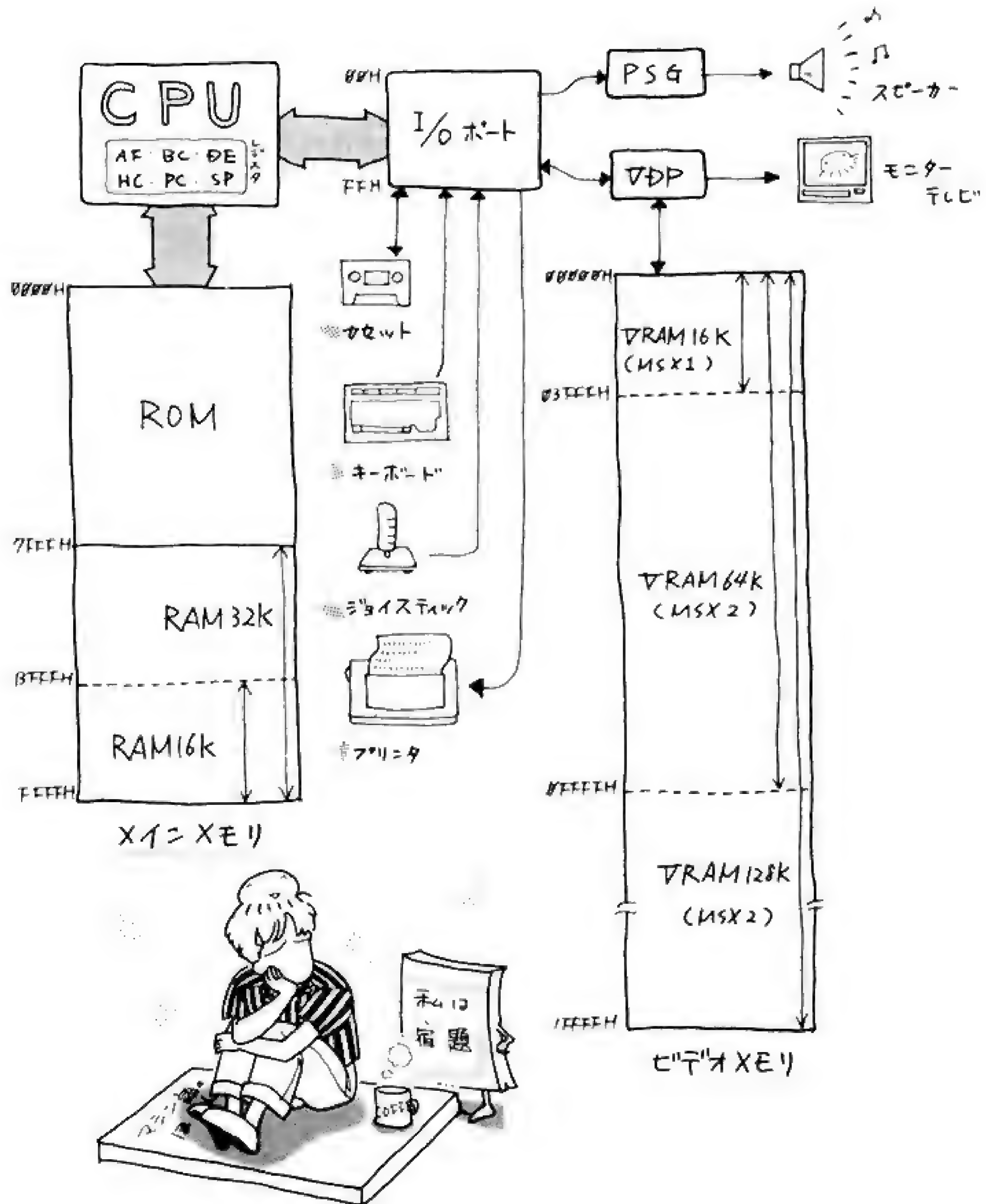


図5-1 MSXの構成

● MSX のハードウェアを探ってみよう

「ハードウェア」とは、コンピュータを構成する装置をいいます。それに対し、プログラムなどのデータを「ソフトウェア」といいます。ここでは、MSX の入出力装置をコントロールするために、前の図を見ながら少しだけハードウェアに触れてみましょう。

MSX などのコンピュータは、CPU（シーピーユー）、メモリ、I/O（アイオー）の3つの部分からできています。

CPUとは「Central Processing Unit」の略で、訳すと「中央演算処理装置」といいます。CPUはコンピュータの中心的存在であり、マシン語を解読して実際の処理に結び付け、実行する所です。さんざん使ってきたレジスタは、この中に収められています。

メモリは、いままでに言ったように、マシン語であつかうデータやマシン語のプログラムなどを記憶しておく部分です。メモリはCPUの指令により、データをCPUに送ったり、CPUから送られてきたデータを記憶したりします。

I/Oとは「Input/Output Unit」の略で、およそすべての入出力装置がつながっている部分です。カセット、キーボード、プリンタ、VDP（画面をコントロールする部分）、PSG（音声をコントロールする部分）といったものは、どれもここを介して接続されています。

I/Oにはアドレスと同じような「ポート番号」が割り当てられており、00H から FFH までの256バイトの空間を持っています。この空間を「I/O ポート」といいます。そして、I/O ポートのいずれかのところに各入出力装置が割り当てられており、装置とのやりとりは、そのポートに対し、データを読み書きすることで行われます。I/O ポートはマシン語ではIN（イン）命令とOUT（アウト）命令で読み書きできます（この命令については、特に覚える必要はありません）。

I/O ポートの状態はBASICのINP文でも見ることもできるので、少し実験してみましょう。カセットにデータの入っているテープをセットして、ロード状態にします（リモート端子は外しておき

ます)。それからリスト 5-1 のプログラムを実行してみてください。

リスト5-1 BASICのプログラム

```

100 '*** カセット からノ シンゴウ ラ カクニン ***
110 '
120 CLEAR 100:CLS
130 LOCATE 0,0
140 A=&HA2
150 D=INP(A)
160 PRINT HEX$(A); "H = ";
170 PRINT RIGHT$("0000000"+BIN$(D),8);
180 PRINT "(";RIGHT$("0"+HEX$(D),2);")"
190 GOTO 130

```

カセットからデータ音が聞こえると、ビット 7 が変化しますね。カセットの信号は、この A2H 番地のビット 7 のポートから入ってくるのです。

他の装置にしても、これほど簡単ではありませんが、やはりポートを介してつながっているわけです。そのため、もし入出力装置を直接コントロールしようと思ったら、マシン語の命令を覚える他に、その装置についてのしっかりした知識も必要になります。

5.2 画面に文字を出してみよう

入出力装置をコントロールするには、本来ならば I/O ポートを介して、データのやりとりをしなければなりません。しかし、これには MSX のハードウェアを詳しく知る必要があり、また MSX の場合、機種や目的によってやり方が少し異なる場合もあるので、かなりたいへんです。そのため MSX には、ハードウェアに対しある程度の知識があればコントロールでき、さらにどの MSX でも必ず動作するようなプログラムが作れるように「BIOS (バイオス)」というサブルーチン群が用意されています。そして、入出力を行うときには、ほとんどの場合 BIOS を使うように定められています。BASIC

インタプリタも、入出力装置をコントロールするとき、BIOS を使っています。ちなみに、BIOS とは「Basic・Input・Output・System」を略したものです。

BIOS の使い方は簡単で、一般的には必要なデータを所定のレジスタに設定し、後は目的に応じた BIOS をコールすればいいことになっています。では、手始めに PRINT 文のように、画面のカーソル位置に文字を表示する BIOS を紹介しましょう。

・画面のカーソル位置に 1 文字表示する BIOS のルーチン

ラベル名	CHPUT
アドレス	00A2H
変化するレジスタ	なし
使い方	A レジスタに表示したいキャラクタコードを設定しコールする

「ラベル名」は、BIOS を作った人が決めた、BIOS ルーチンの名前です。これを EQU 命令で定義して使用すると、他の人にもわかりやすいプログラムを作ることができます。「アドレス」は、そのルーチンの開始番地です。「変化するレジスタ」は、そのルーチ

リスト5-2 プログラムおよび実行例

100	' ; **** 1 モジ" ヒョウシ" ****
110	'
120	' CHPUT EQU 00A2H
130	'
140	' ORG 0D000H
150	'
160	' LD A, 41H
170	' CALL CHPUT
180	'
190	' RET

DEFUSR=&HD000
Ok
A=USR(0)
A表示された文字
Ok

ンをコールして戻ってきたとき、内容が変化しているレジスタの名前です。もし、内容を壊したくないレジスタが変化することになっていた場合は、あらかじめ退避しておきましょう。

それでは、例として文字“A”を表示するプログラムを作ってみます（リスト 5-2 ）。

●文字のあつかい方

マシン語では、文字を 1 バイトの数値に変換してメモリに記憶させています。つまり 1 つの数値は、ひとつの文字（キャラクタ）に対応しているわけです。この数値のことをキャラクタコードと呼び、MSX では表 5-1 のように対応しています。“MSX では”とした

		上 位 4 ビ ッ ト															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
下 位 4 ビ ッ ト	0		π		0	@	P		p	♠			ー	タ	ミ	た	み
	1	月	┐	!	1	A	Q	a	q	♥	あ	。	ア	チ	ム	ち	む
	2	火	┐	"	2	B	R	b	r	♣	い	「	イ	ツ	メ	つ	め
	3	水	┐	#	3	C	S	c	s	♦	う	」	ウ	テ	モ	て	も
	4	木	┐	\$	4	D	T	d	t	○	え	、	エ	ト	ヤ	と	や
	5	金	┐	%	5	E	U	e	u	●	お	。	オ	ナ	ユ	な	ゆ
	6	土	┐	&	6	F	V	f	v	を	か	ヲ	カ	ニ	ヨ	に	よ
	7	日	┐	'	7	G	W	g	w	あ	き	ア	キ	ヌ	ラ	ぬ	ら
	8	年	┐	(8	H	X	h	x	い	く	イ	ク	ネ	リ	ね	り
	9	円	┐)	9	I	Y	i	y	う	け	ウ	ケ	ノ	ル	の	る
	A	時	┐	*	:	J	Z	j	z	え	こ	エ	コ	ハ	レ	は	れ
	B	分	┐	+	;	K	[k	┐	お	さ	オ	サ	ヒ	ロ	ひ	ろ
	C	秒	×	,	<	L	¥	l	┐	ゃ	し	ャ	シ	フ	ワ	ふ	わ
	D	百	大	-	=	M]	m	┐	ゅ	す	ユ	ス	ヘ	ン	へ	ん
	E	千	中	.	>	N	^	n	~	よ	せ	ヨ	セ	ホ	ゝ	ほ	
	F	万	小	/	?	O	—	o		っ	そ	ッ	ソ	マ	。	ま	

③ CHPUTルーチンでは00H～1FHまでがコントロールキャラクタとなっています。

表5-1 キャラクタコード表

のは、他にもいろいろな対応があるためで、他のコンピュータを使う場合に気を付けなければならない点でもあります。

この表の 00H~7FH までのキャラクタコードは、特に「アスキーコード」と呼んでいて、全世界共通に使われています（ただしアスキーコードという名前があまりにも有名であるため、表 5-1 全体のことをアスキーコードとして表現してしまう場合もあります）。

この表によると“A”という文字のキャラクタコードは 41H であることがわかります。また逆にキャラクタコード 42H が表す文字は“B”となります。このコードを VRAM に書き込むことにより、対応する文字が画面に表示されます（ただし VRAM のどこに書いても表示されるわけではありません）。

BASIC で、特定の文字のキャラクタコードや、キャラクタコードに対応する文字を求めるときは、次のようにします。

- ・文字“A”のアスキーコードを 16 進数で求める

```
PRINT HEX$ (ASC ("A "))  
41  
ok
```

- ・キャラクタコードの 43H が表す文字を求める

```
PRINT CHR$ (&H43)  
C  
ok
```

マシン語では、文字そのものをあつかうことができないので、キャラクタコードを使うしかありません。しかし、アセンブリ言語でプログラムを作る場合には、もう少し文字があつかいやすいようにくふうされています。

たとえば“A”のキャラクタコードである 41H をプログラム中に書こうとしたときに、その代わりとして‘A’と書くことができるようになっています。これも、アセンブルしたときには結局 41H というキャラクタコードになって、マシン語に組み込まれます。

アセンブリ言語	アセンブル	マシン語
LD A, 41H	→	3E 41
LD A, 'A'	→	3E 41

● 長い文字列を表示してみよう

さきほどは、とりあえず画面に 1 文字だけ表示してみました。しかし、現実には 1 文字だけ表示することはまれです。そこで、1 文字以上の文字列を表示する方法を考えてみましょう。

1 番わかりやすい方法は、すなおい全部 1 文字出力することです(リスト 5-3)。

リスト5-3 プログラムおよび実行例

```

100 ' ;**** モシ"レ"セウシ" 1 ****
110 '
120 'CHPUT EQU      00A2H
130 '
140 '      ORG      0D000H
150 '
160 '      LD       A, 'K'
170 '      CALL    CHPUT
180 '
190 '      LD       A, 'E'
200 '      CALL    CHPUT
210 '
220 '      LD       A, 'I'
230 '      CALL    CHPUT
240 '
250 '      LD       A, 'K'
260 '      CALL    CHPUT
270 '
280 '      LD       A, 'O'
290 '      CALL    CHPUT
300 '
310 '      RET

```

```

DEF USR=&HD000
Ok
A=USR(0)
KEIKO
Ok

```

この方法は簡単でいいのですが、あまり文字数が多いと、プログラムが大きくなり、作るのがめんどろになります。そこで一般にはリスト 5-4 のような方法がとられます。

リスト5-4 プログラム例

```

100 ' ; **** モシ・レツヒョウシ 2 ****
110 '
120 ' CHPUT EQU      00A2H
130 '
140 '          ORG      0D000H
150 '
160 '          LD       HL, NAME
170 '
180 ' LOOP: LD,        A, (HL)
190 '          CALL     CHPUT
200 '          INC      HL
210 '          CP       0
220 '          JP       NZ, LOOP
230 '
240 '          RET
250 '
260 ' NAME: DEFM      'KEIKO'
270 '          DEFB      00H

```

初め、HLレジスタに文字列の先頭アドレスを設定しておき、文字データが 00H になるまで、1 文字出力を繰り返します。ここででてきた DEFM(デファインメモリ)命令はアセンブラの疑似命令で、アセンブルしたときにオペランドの文字列をキャラクタコードに変換するという働きをします。

● データを設定してみよう

DEFM 命令は、文字データを設定する命令です。このようなデータ設定命令を書いておくと、アセンブルしたとき、このデータは 2 進数の形に変換されそのままメモリに置かれます。

この命令は BASIC でいうと、DATA 文に近いといえます。DATA 文を使うと、データだけまとめて管理できるので、プログラムがスッキリとします。BASIC では、どんなデータも DATA 文で書く

ことができますが、アセンブラで書く場合は、データの形式によって異なり、DEFM 命令も含め以下の 4 種類の疑似命令が用意されています。

① DEFB (デファインバイト) 命令

DEFB n, n, n...

メモリに 8 ビットのデータを設定します。カンマでデータを区切れば、複数個のデータを書くことができます。

② DEFM (デファインメモリ) 命令

DEFM '文字列'

メモリに文字列のデータを設定します、オペランドは 1 つだけで、カンマで区切るようなことはできません。

③ DEFW (デファインワード) 命令

DEFW nn, nn, ...

メモリに 16 ビットのデータを設定します、データは上位 8 ビットと下位 8 ビットが逆にメモリに書かれます。おもに、アドレスとしてのデータを設定する場合に使われます。カンマで区切れば複数個のデータを書くことができます。

④ DEFS (デファインストレージ) 命令

DEFS n

n バイト分のメモリを確保します。

これらの疑似命令をアセンブルしてみましょう (リスト 5-5)。



リスト5-5 アセンブルリスト(プリンタ出力)

100:	;**** DATA ****				
110:					
120:	D000		ORG		0D0000H
130:					
140:	D000	41424144	DAT1:	DEFB	65,42H,'A',68
160:	D004	42414443	DAT2:	DEFW	4142H,'CD'
180:	D008	41424344	DAT3:	DEFM	'ABCD'
200:	D00C		DAT4:	DEFS	8
220:	D014	FFFFFFFF	DAT5:	DEFB	0FFH,0FFH,255
D000	DAT1	D004	DAT2	D008	DAT3
D00C	DAT4	D014	DAT5		

5.3 キーボードから入力してみよう

BASIC の INPUT 文にあたる BIOS には、次のようなものがあります。BASIC の INPUT 文では、長い文字を入力することができますが、これは CHPUT 同様 1 文字しか入力することしかできません。何文字かまとめて入力させたい場合は、このルーチンを何度かコールする必要があります。

・キーボードから 1 文字入力するための BIOS のルーチン	
ラベル名	CHGET
アドレス	009FH
変化するレジスタ	AF
使い方	コールするとキー入力待ちになる。キー入力するとその文字のアスキーコードが A レジスタに入ってくる

CHPUT の場合は変化するレジスタがありませんでしたが、ここでは AF レジスタが変化してしまいます。よくある話ですが、コールする前にフラグを変化させ、コールした後そのフラグをみて分岐するような場合に、コール先でフラグが変わってしまつて、正常に

動作しないことがあります。どこかをコールする前には、必ずレジスタを退避し、堅実なプログラムを作りましょう。

例として、入力した文字をそのまま画面に出すプログラムを作ってみます（リスト 5-6）。

リスト5-6 プログラム例

```

100 ' ; **** モシ ニュウリョク ****
110 '
120 ' CHPUT EQU      00A2H
130 ' CHGET EQU      009FH
140 '
150 '          ORG      0D000H
160 '
170 '          LD        A, 5
180 ' LOOP:
190 '          PUSH      AF
200 '          CALL      CHGET
210 '          CALL      CHPUT
220 '          POP       AF
230 '
240 '          DEC        A
250 '          JP         NZ, LOOP
260 '
270 '          RET

```

5.4 カーソルを移動させてみよう

LOCATE 文に相当する BIOS ルーチンです。指定方法はそれとまったく同じで、X 座標と Y 座標で行います。

・カーソルを移動するための BIOS のルーチン

ラベル名	POSIT
アドレス	00C6H
変化するレジスタ	AF
使い方	Hレジスタに X 座標、L レジスタに Y 座標を入れ、コールする。

P126の文字列を出力するプログラムに、このPOSITルーチンを組み込んでみましょう（リスト5-7）。

リスト5-7 プログラム例

```

100 ' ; **** カーソル位置 ****
110 '
120 ' CHPUT EQU      00A2H
130 ' POSIT EQU      00C6H
140 '
150 '          ORG      0D000H
160 '
170 '          LD        H, 20 ; X ] .....カーソル設定
180 '          LD        L, 10 ; Y ]
190 '          CALL      POSIT
200 '
210 '          LD        HL, NAME
220 '
230 ' LOOP:  LD        A, (HL)
240 '          CALL      CHPUT
250 '          INC        HL
260 '          CP        0
270 '          JP        NZ, LOOP
280 '
290 '          RET
300 '
310 ' NAME:  DEFM      'KEIKO'
320 '          DEFB      00H

```

実行すると、画面中央に文字列が表示されます。

5.5 CtrlとSTOPキーが押されているか調べよう

BASICのプログラムは[Ctrl]と[STOP]キーが同時に押されると、プログラムの実行が中止されます。ところが、マシン語実行中は[Ctrl]と[STOP]キーが押されていても、BASICには戻りません。

[Ctrl]と[STOP]キーが押されたときに、マシン語の実行を中止し、BASICに戻す場合は、キーが押されているかどうかプログラ

ム内で始終みてやらなければなりません。そんなとき、この BIOS ルーチンを使うと便利です。

・ Ctrl と STOP キーが押されているか調べる BIOS ルーチン

ラベル名	BREAKX
アドレス	00B7H
変化するレジスタ	AF
使い方	コールした後、 Ctrl と STOP キーが押されていれば、Cy フラグがセットされ、そうでなければリセットされる。

BREAKX のプログラム例を示します (リスト 5-8)。

結果が Cy フラグでわかるので、次に Cy フラグで分岐する命令を入れておけばいいですね。

リスト5-8 プログラム例

```

100 ' ; **** CTRL+STOP ****
110 '
120 ' BREAKX EQU      00B7H
130 '
140 '          ORG      0D000H
150 '
160 ' LOOP:
170 '          CALL     BREAKX
180 '          JP       NC, LOOP
190 '
200 '          RET

```

このプログラムは、永久ループするプログラムの中に、BREAKX を組み込んだ例です。**Ctrl** と **STOP** キーを押すことによりプログラムが終了します。

5.6 ジョイスティックとトリガボタンの状態を調べよう

ゲームになくてはならないのが、ジョイスティックとトリガボタンですね。BASICにはSTICK関数とSTRIG関数があるので、ジョイスティックがどの方向に押されているか、あるいはトリガボタンが押されているかがすぐにわかります。

ありがたいことに、BIOSにはこれとまったく同じように使える（同じ値を返す）ルーチンがあります。

・ジョイスティックの状態を調べる BIOS ルーチン

ラベル名	GTSTCK
アドレス	00D5H
変化するレジスタ	ぜんぶ
使い方	Aレジスタに知りたいジョイスティック番号(0=カーソル、1=ジョイスティック 1、2=ジョイスティック 2)を設定してからコールする。戻ってきたとき Aレジスタには、押されている方向が BASIC の STICK 関数と同じ値(0~8)で入ってくる。

・トリガボタンの状態を調べる BIOS ルーチン

ラベル名	GTTRIG
アドレス	00D8H
変化するレジスタ	AF
使い方	Aレジスタに知りたいトリガボタン番号(0=スペースキー、1=トリガボタン 1、2=トリガボタン 2)を設定してからコールする。戻ってきたとき、もし Aレジスタが 0 ならば、トリガボタンは押されていない、0 以外ならば押されている。

5.6 ジョイスティックとトリガボタンの状態を調べよう

両ルーチンを使って、画面の左上にジョイスティックの方向とトリガボタンの状態を表示してみます（リスト 5-9）。

リスト5-9 プログラム例

```
100 ' ; **** ジョイスティック ****
110 '
120 ' CHPUT EQU 00A2H
130 ' POSIT EQU 00C6H
140 ' GTSTCK EQU 00D5H
150 ' GTTRIG EQU 00D8H
160 ' BREAKX EQU 00B7H
170 '
180 ' ORG 0D000H
190 ' LOOP:
200 ' CALL BREAKX
210 ' JP C,EXIT
220 ' ; カーソル 設定
230 ' LD HL,0101H
240 ' CALL POSIT
250 ' ; ジョイスティックノ ニューリョク
260 ' LD A,0
270 ' CALL GTSTCK
280 ' ADD A,30H
290 ' CALL CHPUT
300 ' ; カーソル 設定
310 ' LD HL,0102H
320 ' CALL POSIT
330 ' ; トリガ ボタンの ニュウリョク
340 ' LD A,0
350 ' CALL GTTRIG
360 ' CP 0
370 ' JP Z,OFF
380 ' LD A,'1'
390 ' JP PUT
400 ' OFF: LD A,'0'
410 ' PUT: CALL CHPUT
420 '
430 ' JP LOOP
440 ' EXIT:
450 ' RET
```

.....ボタンが押されていないとき

ADD 命令で、GTSTCK ルーチンから戻ってきた数値に 30H を足していますが、これは 0~8 の値に 30H を足すことで、キャラクタコードの "0" ~ "8" に対応させるためのものです。数値を画面に表示するためには、数値に対応するキャラクタコードに変換

しなければなりません。

同じように GTTRIG ルーチンの方も、押されたら文字の "1" を、そうでなければ文字の "0" を画面左上に表示するようにしてみました。

5.7 CAPランプをつけたり、消したりしてみよう

MSX のキーボードなら、どれにでも付いている CAP キーのランプを点滅させてみます。何気なく付いているこのランプですら入出力装置であることを知っておいてください。CAP ランプをコントロールする BIOS を紹介します。

・CAP ランプをつけたり消したりする BIOS ルーチン

ラベル名	CHGCAP
アドレス	0132H
変化するレジスタ	AF
使い方	A レジスタに 0 を入れてコールするとランプが消え、0 以外を入れてコールするとランプがつく

例として、CAP ランプを周期的に点滅させるプログラムを作ってみます(リスト 5-10)。なお、CAP ランプが変化しても、文字の入力モードは変化しません(モードの切り換えと、ランプは全然別のものです)。

リスト5-10 プログラム例

```

100 ' ; **** CAPランプ ****
110 '
120 '      ORG      0D000H
130 '
140 ' CHGCAP EQU    0132H
150 ' BREAKX EQU    00B7H
160 '

```



```
170 ' LOOP:
180 '      CALL      WAIT
190 '
200 ' ;CAPランプ ON
210 '      LD        A,0FFH
220 '      CALL      CHGCAP
230 '
240 '      CALL      WAIT
250 '
260 ' ;CAPランプ OFF
270 '      LD        A,0
280 '      CALL      CHGCAP
290 ' ;CTRL+STOP
300 '      CALL      BREAKX
310 '      JP        NC,LOOP
320 '
330 '      RET
340 '
350 ' ;シゝカンマチ ルーチン
360 ' WAIT:
370 '      PUSH      AF
380 '      PUSH      BC
390 '      PUSH      HL
400 '
410 '      LD        BC,4000H
420 ' LOOP1: LD        HL,0000H
430 '      DEC        BC
440 '      CP         A
450 '      SBC        HL,BC
460 '      JP        NZ,LOOP1
470 '
480 '      POP       HL
490 '      POP       BC
500 '      POP       AF
510 '
520 '      RET
```

時間待ちルーチンでは、16ビットの判定の方法を応用して4000H回（16384回）の無駄ループをさせています。



5.8 VPOKE、VPEEKしてみよう

VRAM の上のデータを読みだしたり、VRAM にデータを書き込んだりするルーチンで、要するに VPOKE、VPEEK とまったく同じ働きをするものです。画面関係は複雑なので、構造を知らないとなかなか難しいのですが、これを使うと CHPUT よりも高速に文字を表示させたりできるので、やりがいがあります。また、スプライトのコントロールなども行うことができます（今回は専門的な説明はしません）。ここでは、SCREENO～3 のモードで利用できるルーチンを紹介しておきます。

- ・ VRAM のデータを読みだす BIOS ルーチン

ラベル名	RDVRM
アドレス	004AH
変化するレジスタ	AF
使い方	HL レジスタに読み出したい VRAM のアドレスを入れ、コールする。戻ってきたとき、A レジスタに読み出したデータが入っている。

- ・ VRAM にデータを書き込む BIOS ルーチン

ラベル名	WRTVRM
アドレス	004DH
変化するレジスタ	AF
使い方	HL レジスタに書き込みたい VRAM のアドレス、A レジスタに書き込みたいデータを入れてコールする。

画面全体にキャラクタを表示するプログラムをあげておきます。
SCREENO：WIDTH40 の状態で実行してください（リスト 5-11）。

リスト5-11 プログラム例

```

100 ' ;**** VPOKE ****
110 '
120 ' WRTVRM EQU      004DH
130 '
140 '          ORG      0D000H
150 '
160 '          LD       B,00H
170 '          LD       DE,0
180 '
190 ' LOOP1: LD        HL,03C0H
200 '
210 ' LOOP2: DEC       HL
220 '          LD       A,B
230 '          CALL    WRTVRM
240 '          SBC      HL,DE
250 '          JP       NZ,LOOP2
260 '
270 '          INC      B
280 '          JP       NZ,LOOP1
290 '
300 '          RET

```

このプログラムは2つのループでできています。1つは画面に対応するVRAMにキャラクタコードを書き込むもので、HLレジスタがカウントしています。もう1つはFFHから00Hまでキャラクタを設定するためのもので、Bレジスタがカウントしています。

1画面が一瞬に表示されるので、非常に高速です。これと全く同じことをする、BASICのプログラムを載せておきますから、比較してみてください（リスト5-12）。

リスト5-12 BASICのプログラム

```

100 ' **** VRAM チョクセツ カキコミ ****
110 '
120 FOR B=&H0 TO &HFF
130   FOR A=&H3C0 TO &H0 STEP -1
140     VPOKE A,B
150   NEXT
160 NEXT

```

BIOS には、他に音を出す SOUND 文のようなルーチン、カセットのデータを読むためのルーチンなどまだまだあります。本シリーズでも後々にまとめたいと思いますが、もし BIOS に興味が出て今すぐ知りたくなった場合は、アスキー出版発行の「MSX2 テクニカルハンドブック」にすべて掲載されているので、そちらを見てくださいね。

付録

•



●マシン語命令表

■8ビットLD(ロード)命令

ニーモニック	マシンコード	フラグ		内 容
		Z	Cy	
LD A,n LD B,n LD C,n LD D,n LD E,n LD H,n LD L,n	3E <u>n</u> 06 <u>n</u> 0E <u>n</u> 16 <u>n</u> 1E <u>n</u> 26 <u>n</u> 2E <u>n</u>	—	—	$r \leftarrow n$
LD A,A LD A,B LD A,C LD A,D LD A,E LD A,H LD A,L	7F 78 79 7A 7B 7C 7D	—	—	$A \leftarrow r$
LD B,A LD B,B LD B,C LD B,D LD B,E LD B,H LD B,L	47 40 41 42 43 44 45	—	—	$B \leftarrow r$
LD C,A LD C,B LD C,C LD C,D LD C,E LD C,H LD C,L	4F 48 49 4A 4B 4C 4D	—	—	$C \leftarrow r$
LD D,A LD D,B LD D,C LD D,D LD D,E LD D,H LD D,L	57 50 51 52 53 54 55	—	—	$D \leftarrow r$
LD E,A LD E,B LD E,C LD E,D LD E,E LD E,H LD E,L	5F 58 59 5A 5B 5C 5D	—	—	$E \leftarrow r$

つづく

ニーモニック	マシンコード	フラグ		内 容
		Z	Cy	
LD H,A LD H,B LD H,C LD H,D LD H,E LD H,H LD H,L	67 60 61 62 63 64 65	—	—	H ← r
LD L,A LD L,B LD L,C LD L,D LD L,E LD L,H LD L,L	6F 68 69 6A 6B 6C 6D			L ← r
LD (nn),A	32 <u>n</u> <u>n</u>			(nn) ← A
L D (BC),A L D (DE),A	02 12			(rr) ← A
LD A,(nn)	3A <u>n</u> <u>n</u>			A ← (nn)
LD A,(BC) LD A,(DE)	0A 1A			A ← (rr)
LD (HL),A LD (HL),B LD (HL),C LD (HL),D LD (HL),E LD (HL),H LD (HL),L	77 70 71 72 73 74 75	—	—	(HL) ← r
LD A,(HL) LD B,(HL) LD C,(HL) LD D,(HL) LD E,(HL) LD H,(HL) LD L,(HL)	7E 46 4E 56 5E 66 6E			r ← (HL)

■16ビットLD(ロード)命令

ニーモニック	マシンコード	フラグ		内 容
		Z	Cy	
LD BC,nn LD DE,nn LD HL,nn	01 <u>n</u> <u>n</u> 11 <u>n</u> <u>n</u> 21 <u>n</u> <u>n</u>	—	—	rr ← nn
LD BC,(nn) LD DE,(nn) LD HL,(nn)	ED 4B <u>n</u> <u>n</u> ED 5B <u>n</u> <u>n</u> 2A <u>n</u> <u>n</u>			rrの下位バイト ← (nn) rrの上位バイト ← (nn+1)
LD (nn),BC LD (nn),DE LD (nn),HL	ED 43 <u>n</u> <u>n</u> ED 53 <u>n</u> <u>n</u> 22 <u>n</u> <u>n</u>			(nn) ← rrの下位バイト (nn+1) ← rrの上位バイト

■PUSH(プッシュ),POP(ポップ)命令

ニーモニック	マシンコード	フラグ		内 容
		Z	Cy	
PUSH AF	F5	—	—	(SP-1) ← rrの上位バイト
PUSH BC	C5			(SP-2) ← rrの下位バイト
PUSH DE	D5			SP ← SP-2
PUSH HL	E5			
POP AF	F1	—	—	rrの下位バイト ← (SP)
POP BC	C1			rrの上位バイト ← (SP+1)
POP DE	D1			SP ← SP+2
POP HL	E1			

■8ビット演算命令

ニーモニック	マシンコード	フラグ		内 容
		Z	Cy	
ADD A, n	C6 <u>n</u>	●	●	Add(アド)
ADD A, A	87			8ビット加算
ADD A, B	80			A ← A + n
ADD A, C	81			A ← A + r
ADD A, D	82			A ← A + (HL)
ADD A, E	83			
ADD A, H	84			
ADD A, L	85			
ADD A, (HL)	86			
SUB n	D6 <u>n</u>	●	●	Subtract(サブトラクト)
SUB A	97			8ビット減算
SUB B	90			A ← A - n
SUB C	91			A ← A - r
SUB D	92			A ← A - (HL)
SUB E	93			
SUB H	94			
SUB L	95			
SUB (HL)	96			
ADC A, n	CE <u>n</u>	●	●	Add with Cy
ADC A, A	8F			(アド・ウィズ・キャリ)
ADC A, B	88			8ビットキャリ付加算
ADC A, C	89			A ← A + n + Cy
ADC A, D	8A			A ← A + r + Cy
ADC A, E	8B			A ← A + (HL) + Cy
ADC A, H	8C			
ADC A, L	8D			
ADC A, (HL)	8E			
SBC A, n	DE <u>n</u>	●	●	Subtract with Cy
SBC A, A	9F			(サブトラクト・ウィズ・キャリ)
SBC A, B	98			8ビットキャリ付減算
SBC A, C	99			A ← A - n - Cy
SBC A, D	9A			A ← A - r - Cy
SBC A, E	9B			A ← A - (HL) - Cy
SBC A, H	9C			
SBC A, L	9D			
SBC A, (HL)	9E			

つづく

ニーモニック	マシンコード	フラグ		内 容
		Z	Cy	
INC A INC B INC C INC D INC E INC H INC L INC (HL)	3C 04 0C 14 1C 24 2C 34	●	—	Increment(インクリメント) 8ビットインクリメント $r \leftarrow r + 1$ $(HL) \leftarrow (HL) + 1$
DEC A DEC B DEC C DEC D DEC E DEC H DEC L DEC (HL)	3D 05 0D 15 1D 25 2D 35	●	—	Decrement(デクリメント) 8ビットデクリメント $r \leftarrow r - 1$ $(HL) \leftarrow (HL) - 1$
CP n CP A CP B CP C CP D CP E CP H CP L CP (HL)	FE <u>n</u> BF B8 B9 BA BB BC BD BE	●	●	Compare(コンペア) 比較 $A - n$ $A - r$ $A - (HL)$

■16ビット演算命令

ニーモニック	マシンコード	フラグ		内 容
		Z	Cy	
ADD HL, BC ADD HL, DE ADD HL, HL	09 19 29	—	●	Add(アド) 16ビット加算 $HL \leftarrow HL + rr$
ADC HL, BC ADC HL, DE ADC HL, HL	ED 4A ED 5A ED 6A	●	●	Add with Cy (アド・ウィズ・キャリ) 16ビットキャリ付施算 $HL \leftarrow HL + rr + Cy$
SBC HL, BC SBC HL, DE SBC HL, HL	ED 42 ED 42 ED 62	●	●	Subtract with Cy (サブトラクト・ウィズ・キャリ) 16ビットキャリ付減算 $HL \leftarrow HL - rr - Cy$
INC BC INC DE INC HL	03 13 23	—	—	Increment(インクリメント) 16ビットインクリメント $sr \leftarrow rr + 1$
DEC BC DEC DE DEC HL	0B 1B 2B	—	—	Decrement(デクリメント) 16ビットデクリメント $rr \leftarrow rr - 1$

■ジャンプ命令

ニーモニック	マシンコード	フラグ		内 容
		Z	Cy	
JP nn	C3 <u>n</u> <u>n</u>	—	—	Jump(ジャンプ) 無条件ジャンプ PC←nn
JP NZ,nn JP Z,nn JP NC,nn JP C,nn	C2 <u>n</u> <u>n</u> CA <u>n</u> <u>n</u> D2 <u>n</u> <u>n</u> DA <u>n</u> <u>n</u>	—	—	Jump(ジャンプ) 条件付ジャンプ ・成立 nn番地へジャンプ PC←nn ・不成立 命令無視
JR e	18 <u>e</u>	—	—	Jump relative (ジャンプ・リラティブ) 無条件相対ジャンプ eバイト先へジャンプ PC←PC+e
JR NZ,e JR Z,e JR NC,e JR C,e	20 <u>e</u> 28 <u>e</u> 30 <u>e</u> 38 <u>e</u>	—	—	Jump relative (ジャンプ・リラティブ) 条件付相対ジャンプ ・成立 eバイト先へジャンプ PC←PC+e ・不成立 命令無視

■CALL命令,RET命令

ニーモニック	マシンコード	フラグ		内 容
		Z	Cy	
CALL nn	CD <u>n</u> <u>n</u>	—	—	Call(コール) 無条件サブルーチンコール PUSH PC,PC←nn
CALL NZ,nn CALL Z,nn CALL NC,nn CALL C,nn	C4 <u>n</u> <u>n</u> CC <u>n</u> <u>n</u> D4 <u>n</u> <u>n</u> DC <u>n</u> <u>n</u>	—	—	Call(コール) 条件付サブルーチンコール ・成立 PUSH PC,PC←nn ・不成立 命令無視
RET	C9	—	—	Return(リターン) 無条件リターン POP PC
RET NZ RET Z RET NC RET C	C0 C8 D0 D8	—	—	Return(リターン) 条件付リターン ・成立 POP PC ・不成立 命令無視

- ▶ nは1バイトのデータを表します。nnは2バイトのデータを表します。
- ▶ フラグの印には次の意味があります。
 - : 結果に従い変化する - : 変化しない
- ▶ 命令には、他にこのようなものがあります。ラベル名を付けるときなどには、注意して下さい。

ADC	ADD	AND	BIT	CALL	CCF	CP	CPD	CPDR	CPI
CPIR	CPL	DAA	DEC	DI	DJNZ	EI	EX	EXX	HALT
IM	IN	INC	IND	INDR	INI	INIR	JP	JR	LD
LDD	LDDR	LDI	LDIR	NEG	NOP	OR	OTDR	OTIR	OUT
OUTD	OUTI	POP	PUSH	RES	RET	RETI	RETN	RL	RLA
RLC	RLCA	RLD	RR	RRA	RRC	RRCA	RRD	RST	SBC
SCF	SET	SLA	SRA	SRL	SUB	XOR			



●マシン語モニタプログラム

```

1000 '=====
1010 '
1020 '    MSX MONITOR  REV 1.0
1030 '
1040 '(MSX1,2 16KRAM 16KVRAM TAPE)
1050 '=====
1060 CLEAR 20,&HEB00:SCREEN 0:WIDTH 40
1070 DEFUSR=&HEB03
1080 FOR A=&HEB00 TO &HF28F STEP 8
1090     CS=0
1100     FOR B=A TO A+7
1110         READ D$:D=VAL("&H"+D$)
1120         CS=CS+D
1130         POKE B,D
1140     NEXT
1150     L=PEEK(&HF6A4)*256+PEEK(&HF6A3)
1160     LOCATE 0,3
1170     PRINT"READING LINE NO. ";L
1180     READ S$
1190     CS$=RIGHT$("00"+HEX$(CS),3)
1200     IF CS$<>S$ THEN GOTO 1230
1210 NEXT
1220 A=USR(0):END
1230 '===== ERROR =====
1240 PRINT L;"*ヨウフキンニ ニュウヨクミスガ* アリマス"
1250 BEEP:END
1260 '===== DATA =====
1270 DATA C3,22,EB,F3,01,05,00,11,2DA
1280 DATA 0D,FE,21,17,EB,ED,B0,21,3EC
1290 DATA FF,EA,22,FA,F2,18,0B,F1,50B
1300 DATA C3,1C,EB,00,E5,CD,22,EB,489
1310 DATA E1,C9,F3,ED,73,8E,F2,31,5AE
1320 DATA 8E,F2,FD,E5,DD,E5,E5,D5,6DE
1330 DATA C5,F5,21,00,EB,22,90,F2,46A
1340 DATA 3A,9A,F1,B7,C2,ED,EE,32,54B
1350 DATA 9D,F1,3D,32,9A,F1,2A,8E,440
1360 DATA F2,22,9B,F1,FB,21,56,EB,4FD
1370 DATA CD,7D,F0,F3,18,17,0D,0A,373
1380 DATA 4D,53,58,20,4D,6F,6E,69,2AB
1390 DATA 74,6F,72,20,20,52,65,76,2C2
1400 DATA 20,31,2E,31,00,2A,8B,F2,254
1410 DATA 22,CA,F1,01,05,00,11,DD,2D1
1420 DATA F1,21,E4,FE,ED,B0,3E,C3,592
1430 DATA 21,9C,EE,32,E4,FE,22,E5,4C6
1440 DATA FE,31,82,F2,AF,32,9D,F1,512
1450 DATA 32,C9,F1,3E,DF,32,D7,F1,503
1460 DATA 32,DA,F1,FB,CD,74,F0,CD,5F6
1470 DATA 2C,EC,78,B7,2B,E3,7E,23,3F3
1480 DATA FE,4C,20,0B,3E,FF,32,9D,381
1490 DATA F1,7E,23,FE,44,20,4C,FE,43E
1500 DATA 44,20,11,7E,FE,53,20,06,26A
1510 DATA 23,3E,FF,32,C9,F1,22,9F,40D

```

1520 DATA F1,C3,FA,EC,22,9F,F1,FE,64A
1530 DATA 53,CA,E8,ED,FE,58,CA,25,537
1540 DATA EF,FE,47,CA,34,EE,FE,42,560
1550 DATA 28,0B,FE,52,CA,B1,F0,2B,419
1560 DATA 22,9F,F1,18,16,F3,01,05,2D9
1570 DATA 00,11,E4,FE,21,DD,F1,ED,4CF
1580 DATA B0,AF,32,9A,F1,ED,7B,9B,51F
1590 DATA F1,FB,C9,AF,32,9D,F1,CD,5F1
1600 DATA 74,F0,3E,3F,CD,A2,00,CD,41D
1610 DATA 6A,F0,2A,9F,F1,7E,B7,28,471
1620 DATA 0A,FE,2C,28,06,CD,A2,00,2D1
1630 DATA 23,18,F2,CD,C0,00,C3,89,406
1640 DATA EB,AF,18,07,3E,2A,CD,A2,390
1650 DATA 00,3E,FF,32,9E,F1,CD,56,421
1660 DATA 01,06,00,21,A1,F1,22,9F,27B
1670 DATA F1,CD,9F,00,FE,61,38,06,3FA
1680 DATA FE,7B,30,02,E6,5F,FE,0D,3FB
1690 DATA 28,6B,FE,1B,28,04,FE,03,2D9
1700 DATA 20,0D,78,B7,28,E3,3E,7F,324
1710 DATA CD,A2,00,10,FB,18,D2,FE,462
1720 DATA 08,20,0D,78,B7,28,D2,3E,29C
1730 DATA 7F,CD,A2,00,05,2B,18,C9,2FF
1740 DATA FE,0C,20,0D,3A,9E,F1,B7,3B7
1750 DATA 28,BF,3E,0C,CD,A2,00,18,2B8
1760 DATA A3,FE,01,20,15,CD,9F,00,343
1770 DATA FE,41,38,0E,FE,60,38,A9,3C4
1780 DATA FE,61,38,A5,FE,7B,30,02,3E7
1790 DATA E6,5F,FE,20,38,9B,FE,7F,4B3
1800 DATA 28,97,4F,77,23,04,78,FE,322
1810 DATA 28,38,04,2B,05,0E,07,79,122
1820 DATA CD,A2,00,18,84,36,00,21,262
1830 DATA A1,F1,C9,DD,2A,9F,F1,0E,500
1840 DATA 00,61,69,DD,7E,00,B7,28,304
1850 DATA 23,FE,2C,28,1D,D6,30,FE,396
1860 DATA 0A,38,0C,D6,07,FE,0A,DA,30D
1870 DATA 03,EC,FE,10,D2,03,EC,29,3E7
1880 DATA 29,29,29,B5,6F,0C,DD,23,2AB
1890 DATA 18,D9,DD,23,47,DD,22,9F,3D6
1900 DATA F1,C9,CD,C3,EC,79,B7,28,58E
1910 DATA 03,22,CA,F1,78,B7,28,0F,346
1920 DATA CD,C3,EC,79,B7,CA,03,EC,565
1930 DATA 78,B7,C2,03,EC,18,07,2A,329
1940 DATA CA,F1,11,7F,00,19,22,CC,352
1950 DATA F1,3A,9D,F1,B7,C4,74,F0,598
1960 DATA 06,00,78,B7,20,13,DD,21,266
1970 DATA CE,F1,DD,36,00,00,CD,74,413
1980 DATA F0,2A,CA,F1,CD,86,F0,0E,526
1990 DATA 00,CD,6A,F0,2A,CA,F1,7E,48A
2000 DATA CD,8B,F0,7E,B1,4F,7E,FE,512
2010 DATA 20,38,08,FE,7F,28,04,FE,307
2020 DATA FF,20,02,3E,2E,DD,77,00,2E1
2030 DATA DD,23,DD,36,00,00,23,22,258
2040 DATA CA,F1,2B,04,ED,5B,CC,F1,4EF
2050 DATA B7,ED,52,30,2A,78,FE,08,3CE
2060 DATA 20,C7,CD,B4,ED,CD,9C,00,4BE

2070 DATA 28,A6,CD,9F,00,FE,0D,CA,40F
2080 DATA 89,EB,CD,56,01,CD,9C,00,401
2090 DATA 28,FB,CD,9F,00,FE,0D,CA,464
2100 DATA 89,EB,CD,56,01,18,89,78,3B1
2110 DATA FE,08,28,0A,C5,06,03,CD,2D3
2120 DATA 6E,F0,C1,04,18,F1,CD,B4,4AD
2130 DATA ED,C3,89,EB,3A,C9,F1,B7,5CF
2140 DATA 28,0F,CD,6A,F0,3E,3A,CD,3A3
2150 DATA 9E,F0,CD,6A,F0,79,C3,8B,57C
2160 DATA F0,3A,9D,F1,B7,20,10,3A,3D9
2170 DATA B0,F3,FE,25,DB,20,08,CD,493
2180 DATA DF,ED,3E,08,C3,A2,00,CD,444
2190 DATA 6A,F0,21,CE,F1,C3,7D,F0,56A
2200 DATA CD,C3,EC,79,B7,CA,03,EC,565
2210 DATA 78,B7,C2,03,EC,22,CC,F1,4BF
2220 DATA CD,74,F0,CD,86,F0,CD,6A,5AB
2230 DATA F0,7E,CD,8B,F0,CD,6A,F0,5DD
2240 DATA CD,29,EC,78,B7,28,19,3D,38F
2250 DATA 20,0A,7E,FE,2E,CA,89,EB,412
2260 DATA FE,5E,28,12,CD,C3,EC,B7,4C9
2270 DATA C2,03,EC,7D,2A,CC,F1,77,48C
2280 DATA 2A,CC,F1,23,18,C7,2A,CC,3DF
2290 DATA F1,2B,18,C1,CD,C3,EC,ED,55E
2300 DATA 5B,90,F2,79,B7,20,01,EB,419
2310 DATA 22,CC,F1,78,B7,28,2F,CD,432
2320 DATA C3,EC,79,B7,CA,03,EC,E5,57D
2330 DATA 78,B7,28,18,CD,C3,EC,79,464
2340 DATA B7,CA,03,EC,E5,78,B7,C2,546
2350 DATA 03,EC,E1,22,DB,F1,7E,32,46E
2360 DATA DA,F1,36,DF,E1,22,DB,F1,5AC
2370 DATA 7E,32,D7,F1,36,DF,2A,CC,483
2380 DATA F1,22,90,F2,F3,31,82,F2,52D
2390 DATA ED,5B,90,F2,2A,8E,F2,2B,49F
2400 DATA 72,2B,73,22,8E,F2,F1,C1,464
2410 DATA D1,E1,DD,E1,FD,E1,ED,78,6B6
2420 DATA 8E,F2,FB,C9,E5,D5,21,08,527
2430 DATA 00,39,5E,23,56,1B,3A,D7,23C
2440 DATA F1,FE,DF,28,17,2A,DB,F1,500
2450 DATA B7,ED,52,28,14,3A,DA,F1,437
2460 DATA FE,DF,28,08,2A,DB,F1,B7,4BA
2470 DATA ED,52,28,05,D1,E1,C3,DD,4BE
2480 DATA F1,F3,D1,E1,F1,F1,ED,73,6D8
2490 DATA 8E,F2,31,8E,F2,FD,E5,DD,5F0
2500 DATA E5,E5,D5,C5,F5,2A,8E,F2,603
2510 DATA 5E,23,56,23,22,8E,F2,1B,2B7
2520 DATA ED,53,90,F2,FB,3A,D7,F1,5BF
2530 DATA FE,DF,28,0F,2A,DB,F1,77,47E
2540 DATA 3A,DA,F1,FE,DF,28,04,2A,438
2550 DATA DB,F1,77,AF,32,9D,F1,21,4D3
2560 DATA 1C,EF,CD,7D,F0,2A,90,F2,4F1
2570 DATA 22,CA,F1,CD,86,F0,CD,74,561
2580 DATA F0,C3,B4,EF,0D,0A,42,72,421
2590 DATA 65,61,6B,20,00,2A,9F,F1,30B
2600 DATA 7E,B7,CA,B4,EF,1E,20,57,437
2610 DATA 23,7E,B7,28,07,5F,23,7E,287

2620	DATA	B7,C2,03,EC,01,00,10,21,29A
2630	DATA	3A,F0,7E,BA,23,20,04,7E,327
2640	DATA	BB,28,08,23,23,0C,10,F2,23F
2650	DATA	C3,03,EC,2B,CD,74,F0,CD,4DB
2660	DATA	7D,F0,3E,3D,CD,A2,00,06,35D
2670	DATA	00,79,FE,0B,30,21,21,82,273
2680	DATA	F2,09,7E,CD,8B,F0,E5,CD,573
2690	DATA	6A,F0,CD,29,EC,78,B7,CA,535
2700	DATA	89,EB,CD,C3,EC,B7,C2,03,56C
2710	DATA	EC,7D,E1,77,C3,89,EB,D6,5CE
2720	DATA	0B,87,4F,21,82,F2,09,23,29F
2730	DATA	7E,CD,8B,F0,2B,E5,7E,CD,521
2740	DATA	8B,F0,CD,6A,F0,CD,29,EC,5B4
2750	DATA	78,B7,CA,89,EB,CD,C3,EC,5E9
2760	DATA	B7,C2,03,EC,EB,E1,73,23,4CA
2770	DATA	72,C3,89,EB,CD,74,F0,06,4E0
2780	DATA	0E,CD,6E,F0,21,31,F0,CD,448
2790	DATA	7D,F0,CD,74,F0,11,3D,F0,4DC
2800	DATA	CD,0E,F0,2A,82,F2,E5,7C,4CA
2810	DATA	CD,8B,F0,06,03,CD,6E,F0,47C
2820	DATA	11,3A,F0,CD,0E,F0,E1,7D,464
2830	DATA	CD,8B,F0,3E,28,CD,A2,00,41D
2840	DATA	06,0B,26,18,29,7C,CD,A2,260
2850	DATA	00,10,F7,3E,29,CD,A2,00,2DD
2860	DATA	CD,74,F0,11,55,F0,21,84,42C
2870	DATA	F2,06,03,CD,19,F0,06,04,2DB
2880	DATA	CD,19,F0,C3,89,EB,EB,CD,5C5
2890	DATA	7D,F0,EB,13,3E,3D,C3,A2,44B
2900	DATA	00,CD,0E,F0,D5,5E,23,56,377
2910	DATA	23,EB,CD,86,F0,EB,D1,10,51D
2920	DATA	03,C3,74,F0,CD,6A,F0,18,469
2930	DATA	E8,53,5A,20,48,20,50,4E,2BB
2940	DATA	43,00,46,20,00,41,20,00,10A
2950	DATA	43,20,00,42,20,00,45,20,12A
2960	DATA	00,44,20,00,4C,20,00,48,118
2970	DATA	20,00,41,46,00,42,43,00,12C
2980	DATA	44,45,00,48,4C,00,49,58,1BE
2990	DATA	00,49,59,00,53,50,00,50,195
3000	DATA	43,00,3E,20,18,30,CD,6A,220
3010	DATA	F0,10,FB,C9,3E,0D,CD,9E,47A
3020	DATA	F0,3E,0A,18,21,7E,B7,CB,36E
3030	DATA	CD,9E,F0,23,18,F7,7C,CD,4D6
3040	DATA	8B,F0,7D,F5,0F,0F,0F,0F,329
3050	DATA	CD,94,F0,F1,E6,0F,FE,0A,53F
3060	DATA	38,02,C6,07,C6,30,F5,3A,32C
3070	DATA	9D,F1,B7,20,04,F1,C3,A2,4BF
3080	DATA	00,F1,CD,A5,00,DA,89,EB,4B1
3090	DATA	C9,CD,C3,EC,78,B7,C2,03,539
3100	DATA	EC,E5,FD,E1,3A,AF,FC,B7,64B
3110	DATA	20,32,21,EA,F0,CD,7D,F0,487
3120	DATA	2A,4A,FC,CD,86,F0,3E,2D,41E
3130	DATA	CD,9E,F0,2A,FA,F2,CD,86,5C4
3140	DATA	F0,CD,6A,F0,3E,29,CD,9E,4E9
3150	DATA	F0,CD,74,F0,CD,1A,F1,C3,5BC
3160	DATA	89,EB,0D,0A,46,72,65,65,30D

3170 DATA 20,20,20,00,21,FD,F0,CD,343
3180 DATA 7D,F0,C3,89,EB,0D,0A,53,40E
3190 DATA 63,72,65,65,6E,20,6E,6F,30A
3200 DATA 74,20,34,30,58,32,34,20,1D6
3210 DATA 74,65,78,74,20,6D,6F,64,325
3220 DATA 65,00,21,00,20,18,0A,EB,1B3
3230 DATA 3E,2D,CD,9E,F0,CD,86,F0,509
3240 DATA EB,CD,4A,00,23,5F,CD,4A,39B
3250 DATA 00,23,57,CD,4A,00,23,4F,203
3260 DATA CD,4A,00,23,47,B1,B2,B3,397
3270 DATA C8,E5,FD,E5,E1,19,EB,21,595
3280 DATA 75,F1,CD,7D,F0,EB,CD,86,5DE
3290 DATA F0,EB,E1,78,B1,28,C8,E5,5BA
3300 DATA 2A,4A,FC,2B,B7,ED,52,E1,472
3310 DATA 30,1B,E5,2A,FA,F2,B7,ED,4EA
3320 DATA 52,E1,38,11,CD,4A,00,12,2A5
3330 DATA 13,23,0B,1B,DE,0D,0A,4C,19A
3340 DATA 6F,61,64,20,00,CD,74,F0,385
3350 DATA EB,CD,86,F0,21,8D,F1,CD,59A
3360 DATA 7D,F0,C3,89,EB,20,20,4C,430
3370 DATA 6F,61,64,20,65,72,72,6F,30C
3380 DATA 72,00,00,00,00,00,00,00,072
3390 DATA 00,00,00,00,00,00,00,00,000
3400 DATA 00,00,00,00,00,00,00,00,000
3410 DATA 00,00,00,00,00,00,00,00,000
3420 DATA 00,00,00,00,00,00,00,00,000
3430 DATA 00,00,00,00,00,00,00,00,000
3440 DATA 00,00,00,00,00,00,00,00,000
3450 DATA 00,00,00,00,00,00,00,00,000
3460 DATA 00,00,00,00,00,00,00,00,000
3470 DATA 00,00,00,00,00,00,00,00,000
3480 DATA 00,00,00,00,00,00,00,00,000
3490 DATA 00,00,00,00,00,00,00,00,000
3500 DATA 00,00,00,00,00,00,00,00,000
3510 DATA 00,00,00,00,00,00,00,00,000
3520 DATA 00,00,00,00,00,00,00,00,000
3530 DATA 00,00,00,00,00,00,00,00,000
3540 DATA 00,00,00,00,00,00,00,00,000
3550 DATA 00,00,00,00,00,00,00,00,000
3560 DATA 00,00,00,00,00,00,00,00,000
3570 DATA 00,00,00,00,00,00,00,00,000
3580 DATA 00,00,00,00,00,00,00,00,000
3590 DATA 00,00,00,00,00,00,00,00,000
3600 DATA 00,00,00,00,00,00,00,00,000
3610 DATA 00,00,00,00,00,00,00,00,000
3620 DATA 00,00,00,00,00,00,00,00,000
3630 DATA 00,00,00,00,00,00,00,00,000
3640 DATA 00,00,00,00,00,00,00,00,000
3650 DATA 00,00,00,00,00,00,00,00,000
3660 DATA 00,00,00,00,00,00,00,00,000
3670 DATA 00,00,00,00,00,00,00,00,000
3680 DATA 00,00,00,00,00,00,00,00,000

●アセンブラ・ダンプリスト

D400	C3	2B	D4	C3	18	D4	C3	E5	:	19	D5D0	FE	3A	C8	FE	20	C0	18	F4	:	EA
D408	D5	C3	2E	D6	C3	96	D8	C3	:	90	D5D8	32	FD	F2	78	32	FE	F2	79	:	34
D410	8C	D8	C3	25	D4	C3	25	D4	:	DC	D5E0	32	FF	F2	AF	01	3E	01	32	:	44
D418	F3	01	05	00	11	0D	FE	21	:	36	D5E8	FC	F2	ED	73	76	EA	31	F0	:	CF
D420	26	D4	ED	B0	FB	C9	C3	00	:	1E	D5F0	F2	21	78	EA	01	25	00	36	:	D1
D428	D4	00	00	7E	FE	4D	28	0A	:	CF	D5F8	00	23	0B	78	B1	20	F8	3D	:	AC
D430	FE	41	28	1C	FE	53	CA	B5	:	53	D600	3D	32	00	EA	3A	AF	FC	B7	:	75
D438	D4	C9	23	7E	FE	95	C0	CD	:	5E	D608	C2	DE	D7	21	00	10	01	00	:	A9
D440	CC	D5	C0	F1	E5	21	FF	D3	:	2A	D610	30	AF	CD	56	00	21	FE	D6	:	F7
D448	22	FA	F2	CD	00	EB	E1	C9	:	70	D618	CD	01	DB	3E	01	21	1C	D7	:	FC
D450	23	7E	FE	53	C0	23	7E	FE	:	51	D620	CD	4C	D7	3A	FC	F2	B7	3E	:	0D
D458	4D	C0	16	01	42	4A	CD	CC	:	49	D628	01	C2	F6	D6	18	07	ED	73	:	0E
D460	D5	28	28	FE	22	C0	CD	93	:	65	D630	76	EA	31	F0	F2	CD	B5	D9	:	CE
D468	D4	30	17	57	CD	93	D4	30	:	D6	D638	3E	02	21	26	D7	CD	4C	D7	:	4E
D470	11	47	CD	93	D4	30	0B	4F	:	16	D640	CD	F8	DA	21	30	D7	CD	01	:	95
D478	23	7E	B7	28	0E	FE	22	C0	:	6E	D648	DB	11	B3	EA	2A	AC	EA	CD	:	E6
D480	18	05	7E	FE	22	20	04	CD	:	AC	D650	FB	E0	AF	12	21	B3	EA	CD	:	F7
D488	CC	D5	C0	F1	7A	E5	CD	D8	:	56	D658	01	DB	21	3F	D7	CD	01	DB	:	BC
D490	D5	E1	C9	23	7E	B7	C8	FE	:	9D	D660	11	B3	EA	2A	99	EA	CD	C2	:	BA
D498	3A	C8	FE	22	C8	CD	AC	DC	:	3F	D668	E0	AF	12	21	B3	EA	CD	01	:	FD
D4A0	1E	00	FE	58	28	0C	1C	FE	:	C2	D670	DB	21	42	D7	CD	01	DB	CD	:	BB
D4A8	43	28	07	1C	FE	50	28	02	:	06	D678	F8	DA	3A	80	EA	B7	28	0C	:	61
D4B0	F1	C9	7B	37	C9	CD	CC	D5	:	A3	D680	3C	3C	28	08	3E	0C	CD	A5	:	64
D4B8	FE	22	C0	23	22	74	EA	06	:	89	D688	00	DA	96	D7	CD	F8	DA	3A	:	20
D4C0	00	7E	B7	28	0F	FE	22	28	:	B4	D690	FE	F2	B7	28	5F	21	00	00	:	4F
D4C8	07	23	04	CB	78	28	F2	C9	:	54	D698	ED	5B	99	EA	E5	B7	ED	52	:	A6
D4D0	CD	CC	D5	C0	78	32	73	EA	:	35	D6A0	E1	30	3E	E5	CD	B4	DA	2A	:	B9
D4D8	F1	E5	2A	76	F6	22	7E	EA	:	F6	D6A8	8F	EA	11	B3	EA	CD	FB	E0	:	9F
D4E0	21	5D	D5	CD	BD	D5	0E	00	:	C0	D6B0	3E	20	EB	77	23	77	21	8F	:	0A
D4E8	CD	B7	00	38	6B	C5	21	54	:	61	D6B8	EA	06	04	77	23	10	FC	36	:	D0
D4F0	D5	E5	ED	73	76	EA	CD	1C	:	63	D6C0	00	21	B3	EA	3A	FE	F2	3D	:	F5
D4F8	D8	C1	C1	38	58	C5	22	71	:	42	D6C8	20	08	CD	F8	DA	CD	01	DB	:	70
D500	EA	06	00	7E	B7	28	04	23	:	74	D6D0	18	0B	D1	D5	7B	E6	03	CC	:	F9
D508	04	18	F8	78	32	70	EA	D5	:	ED	D6D8	E3	DA	CD	EC	DA	E1	23	18	:	6C
D510	CD	80	D5	D1	C1	B7	28	D0	:	63	D6E0	B7	3A	FE	F2	3D	20	05	CD	:	10
D518	79	E6	03	CC	B4	D5	C5	EB	:	67	D6E8	F8	DA	18	08	CD	E3	DA	3E	:	BA
D520	11	B3	EA	CD	C2	E0	AF	12	:	AE	D6F0	0C	CD	A5	00	3E	02	32	F9	:	E9
D528	21	B3	EA	CD	BD	D5	21	C6	:	D4	D6F8	F2	ED	7B	76	EA	C9	0D	0A	:	9A
D530	D5	CD	BD	D5	C1	0C	CD	9C	:	6A	D700	4D	53	5B	20	53	65	6C	66	:	A2
D538	00	28	AD	CD	9F	00	FE	20	:	5F	D708	20	41	73	73	65	6D	62	6C	:	E7
D540	20	A6	CD	56	01	CD	B7	00	:	6E	D710	65	72	20	20	52	65	76	20	:	64
D548	38	0E	CD	9C	00	28	F6	CD	:	9A	D718	31	2E	31	00	0D	0A	0A	50	:	01
D550	56	01	18	94	E1	CD	B4	D5	:	3A	D720	61	73	73	20	31	00	0D	0A	:	AF
D558	CD	B4	D5	E1	C9	0D	0A	41	:	58	D728	0A	50	61	73	73	20	32	00	:	F3
D560	73	73	65	6D	62	6C	65	72	:	5D	D730	0D	0A	45	6E	64	20	41	64	:	F3
D568	20	73	6F	75	72	63	65	20	:	D1	D738	64	72	65	73	73	20	00	20	:	61
D570	73	74	72	69	6E	67	20	73	:	2A	D740	2C	00	20	4C	61	62	65	6C	:	2C
D578	65	61	72	63	68	0D	0A	00	:	1A	D748	28	73	29	00	32	9D	EA	CD	:	4A
D580	3A	73	EA	B7	C8	2A	74	EA	:	9E	D750	01	DB	2A	76	F6	22	7E	EA	:	FC
D588	4F	3A	70	EA	B7	C8	ED	5B	:	AA	D758	CD	2C	DB	CD	F8	DA	2A	D6	:	73
D590	71	EA	91	47	3E	00	D8	04	:	4D	D760	EA	7C	B5	F5	28	12	E5	CD	:	FC
D598	3C	F5	C5	D5	E5	1A	BE	20	:	A8	D768	F8	DA	E1	11	B3	EA	CD	C2	:	C0
D5A0	0A	13	23	0D	20	F7	E1	D1	:	16	D770	E0	AF	12	21	B3	EA	18	03	:	4A
D5A8	C1	F1	C9	E1	D1	C1	F1	13	:	F2	D778	21	B9	D7	CD	01	DB	21	8C	:	D7
D5B0	10	E6	AF	C9	E5	21	C9	D5	:	12	D780	D7	CD	01	DB	F1	C8	C3	06	:	02
D5B8	CD	BD	D5	E1	C9	7E	B7	C8	:	06	D788	D8	4E	6F	00	20	45	72	72	:	DE
D5C0	CD	A2	00	23	18	F7	20	20	:	E1	D790	6F	72	28	73	29	00	21	9C	:	62
D5C8	00	0D	0A	00	23	7E	B7	C8	:	37	D798	D7	C3	03	D8	0D	0A	25	20	:	D1

D7A0	41	62	6F	72	74	65	64	00	:	C1	D990	45	00	E5	CD	3C	DA	E1	30	:	1E
D7A8	21	AE	D7	C3	03	D8	0D	0A	:	5B	D998	19	11	06	00	19	ED	5B	8F	:	20
D7B0	25	20	4F	62	6A	65	63	74	:	9C	D9A0	EA	73	23	72	AF	C9	E5	CD	:	1C
D7B8	20	61	72	65	61	20	66	75	:	B4	D9A8	3C	DA	E1	38	05	CD	5E	DA	:	39
D7C0	6C	6C	00	21	C9	D7	C3	03	:	5F	D9B0	AF	C9	F6	FF	C9	2A	99	EA	:	E3
D7C8	D8	0D	0A	25	20	4C	61	62	:	43	D9B8	7C	B5	C8	2B	7C	B5	C8	01	:	1E
D7D0	65	6C	20	74	61	62	6C	65	:	F9	D9C0	00	00	50	59	C5	D5	C5	EB	:	F3
D7D8	20	66	75	6C	6C	00	21	E4	:	D8	D9C8	11	91	EA	CD	87	DA	C1	D1	:	4C
D7E0	D7	C3	03	D8	0D	0A	25	20	:	D1	D9D0	E1	23	C5	D5	E5	CD	84	DA	:	AE
D7E8	53	63	72	65	65	6E	20	6E	:	EE	D9D8	11	91	EA	21	89	EA	CD	75	:	62
D7F0	6F	74	20	34	30	58	32	34	:	25	D9E0	DA	E1	D1	28	13	38	11	54	:	64
D7F8	20	74	65	78	74	20	6D	6F	:	E1	D9E8	5D	D5	E5	01	08	00	11	91	:	C2
D800	64	65	00	CD	01	DB	CD	F8	:	37	D9F0	EA	21	89	EA	ED	B0	E1	D1	:	CD
D808	DA	3E	FF	C3	F6	D6	3A	FC	:	DC	D9F8	23	C1	D5	E5	EB	2A	99	EA	:	36
D810	F2	B7	28	08	ED	73	9B	EA	:	BE	DA00	2B	B7	ED	52	E1	D1	30	CA	:	CD
D818	AF	C3	F6	D6	CD	B7	00	DA	:	9C	DA08	D5	EB	B7	ED	42	D1	2B	1F	:	BE
D820	96	D7	2A	7E	EA	5E	23	56	:	D6	DA10	C5	D5	C5	EB	11	91	EA	CD	:	A3
D828	23	ED	53	7E	EA	7A	B3	28	:	20	DA18	87	DA	E1	E5	CD	84	DA	E1	:	33
D830	24	5E	23	56	D5	11	57	D8	:	10	DA20	E3	11	89	EA	CD	94	DA	E1	:	B3
D838	01	20	03	23	7E	B7	CA	5A	:	A0	DA28	11	91	EA	CD	94	DA	C1	03	:	88
D840	D8	B9	28	F7	0E	00	1A	BE	:	96	DA30	2A	99	EA	2B	2B	B7	ED	42	:	E9
D848	C2	5A	D8	13	23	10	ED	D1	:	F8	DA38	D2	C2	D9	C9	EB	01	00	00	:	22
D850	22	7C	EA	B7	C9	37	C9	3A	:	42	DA40	2A	99	EA	B7	ED	42	C8	60	:	BB
D858	8F	E6	CD	F8	DA	E1	11	83	:	89	DA48	69	C5	D5	CD	84	DA	D1	21	:	20
D860	EA	CD	C2	E0	AF	12	CD	01	:	E8	DA50	89	EA	CD	75	DA	28	04	C1	:	7C
D868	DB	21	6E	D8	18	95	3A	20	:	49	DA58	03	10	E5	C1	37	C9	E5	2A	:	D0
D870	20	20	20	25	20	41	73	73	:	CC	DA60	99	EA	E5	11	00	02	B7	ED	:	1F
D878	65	6D	62	6C	65	72	20	73	:	0A	DA68	52	E1	D2	C3	D7	23	22	99	:	7D
D880	6F	75	72	63	65	20	65	72	:	15	DA70	EA	2B	D1	18	1F	D5	E5	06	:	DD
D888	72	6F	72	00	EB	23	5E	23	:	E2	DA78	06	1A	96	20	04	23	13	10	:	20
D890	56	ED	53	7C	EA	C9	ED	73	:	25	DA80	F8	E1	D1	C9	11	89	EA	29	:	20
D898	76	EA	ED	7B	9B	EA	23	23	:	93	DA88	29	29	01	00	10	09	01	08	:	75
D8A0	5E	23	56	7A	B3	28	AE	2A	:	04	DA90	00	C3	59	00	29	29	29	01	:	98
D8A8	7C	EA	B7	C9	2A	78	EA	23	:	95	DA98	00	10	09	EB	01	08	00	C3	:	D0
D8B0	23	E5	F5	CD	A9	DA	F1	E1	:	1F	DAA0	5C	00	11	00	20	19	C3	4A	:	B3
D8B8	E5	23	23	D5	19	23	22	7A	:	D8	DAA8	00	11	00	20	19	CD	4A	00	:	61
D8C0	EA	2B	CD	B7	DA	D1	E1	13	:	3B	DAB0	5F	23	CD	4A	00	57	C9	E5	:	9E
D8C8	C3	C9	DA	B7	28	0C	7A	B3	:	7E	DAB8	11	FC	1F	B7	ED	52	E1	D2	:	D5
D8D0	C8	D5	AF	CD	AC	D8	D1	1B	:	89	DAC0	A8	D7	11	00	20	19	C3	4D	:	D9
D8D8	18	F4	E5	2A	78	EA	23	23	:	C3	DAC8	00	D5	E5	11	FC	1F	B7	ED	:	BA
D8E0	CD	A9	DA	7A	B3	28	0D	2A	:	DC	DAD0	52	E1	D2	A8	D7	11	00	20	:	B5
D8E8	7A	EA	22	78	EA	23	23	23	:	51	DAD8	19	D1	78	CD	4D	00	23	7A	:	1C
D8F0	23	22	7A	EA	D1	2A	78	EA	:	06	DAE0	C3	4D	00	E5	21	29	DB	CD	:	E7
D8F8	E5	CD	C9	DA	E1	23	23	11	:	8D	DAE8	EC	DA	E1	C9	7E	B7	C8	CD	:	3A
D900	00	00	C3	C9	DA	3A	FD	F2	:	8F	DAF0	A5	00	DA	96	D7	23	18	F4	:	1B
D908	B7	C8	3D	20	10	11	08	00	:	05	DAF8	E5	21	29	DB	CD	01	DB	E1	:	94
D910	19	CD	F8	DA	CD	01	DB	2A	:	8B	DB00	C9	7E	B7	C8	CD	A2	00	23	:	5B
D918	7C	EA	C3	01	DB	E5	CD	40	:	F7	DB08	CD	9C	00	2B	F4	CD	9F	00	:	F1
D920	D9	E1	CD	E3	DA	CD	EC	DA	:	D7	DB10	FE	20	20	ED	E5	CD	56	01	:	34
D928	2A	7C	EA	C3	EC	DA	47	3A	:	9A	DB18	CD	B7	00	DA	96	D7	CD	9C	:	34
D930	FF	F2	FE	02	DA	01	DB	04	:	AB	DB20	00	28	F5	CD	56	01	E1	18	:	3A
D938	E5	CC	40	D9	E1	C3	EC	DA	:	34	DB28	D8	0D	0A	00	ED	73	9F	EA	:	D8
D940	21	80	EA	34	7E	FE	32	D8	:	45	DB30	AF	32	9E	EA	67	6F	22	AC	:	0D
D948	36	00	2A	81	EA	7C	B5	3E	:	3A	DB38	EA	22	D6	EA	ED	7B	9F	EA	:	BD
D950	0C	C4	A5	00	DA	96	D7	21	:	DD	DB40	3A	9D	EA	3D	28	0C	21	B1	:	04
D958	84	D9	CD	EC	DA	21	00	D7	:	E8	DB48	EA	06	18	3E	20	77	23	10	:	10
D960	CD	EC	DA	21	89	D9	CD	EC	:	CF	DB50	FC	70	AF	32	AE	EA	CD	0E	:	C0
D968	DA	2A	81	EA	23	22	81	EA	:	1F	DB58	D8	DA	86	DC	ED	53	AF	EA	:	ED
D970	11	83	EA	CD	C2	E0	AF	12	:	AE	DB60	CD	8C	DC	CA	57	DC	22	AA	:	FE
D978	21	83	EA	CD	EC	DA	CD	E3	:	D1	DB68	EA	7E	CD	AC	DC	CD	A0	DC	:	06
D980	DA	C3	E3	DA	20	20	20	20	:	DA	DB70	DA	62	E9	3A	9D	EA	3D	28	:	4B
D988	00	20	20	20	20	50	41	47	:	5B	DB78	09	2A	AC	EA	11	BB	EA	CD	:	4C

DB80	FB	E0	CD	16	E1	CD	CF	DC	:	17	DD70	45	46	53	86	45	46	57	85	:	CB
DB88	B7	20	38	2A	AA	EA	7E	23	:	6E	DD78	49	20	20	47	4A	4E	5A	1C	:	DE
DB90	FE	3A	28	06	CD	48	E1	C3	:	1F	DD80	49	20	20	48	4E	44	20	81	:	04
DB98	57	DC	22	AA	EA	2A	AC	EA	:	A9	DD88	51	55	20	82	58	20	20	04	:	E4
DBA0	CD	C6	E8	2A	AA	EA	CD	8C	:	92	DD90	58	58	20	40	41	4C	54	46	:	37
DBA8	DC	CA	57	DC	22	AA	EA	7E	:	0D	DD98	4D	20	20	0F	4E	20	20	20	:	4A
DBB0	CD	AC	DC	CD	A0	DC	DA	5F	:	D7	DDA0	4E	43	20	0D	4E	44	20	5F	:	CF
DBB8	E9	CD	16	E1	CD	CF	DC	B7	:	DC	DDA8	4E	44	52	60	4E	49	20	5D	:	58
DBC0	CA	5F	E9	F5	2A	AA	EA	7E	:	43	ddb0	4E	49	52	5E	50	20	20	1A	:	F1
DBC8	FE	3A	CA	62	E9	B7	28	0F	:	3B	ddb8	52	20	20	1B	44	20	20	01	:	32
DBD0	FE	3B	28	0B	FE	20	C2	5F	:	AB	ddc0	44	44	20	52	44	44	52	53	:	27
DBD8	E9	CD	8C	DC	22	AA	EA	F1	:	C5	ddc8	44	49	20	50	44	49	52	51	:	2D
DBE0	21	57	DC	E5	FE	80	38	15	:	04	ddd0	45	47	20	58	4F	50	20	45	:	08
DBE8	D6	80	21	EF	DB	18	1C	A7	:	1C	ddd8	52	20	20	0A	52	47	20	80	:	D5
DBF0	E1	C6	E1	5F	E9	CF	E1	03	:	83	dde0	54	44	52	64	54	49	52	62	:	9F
DBF8	E2	E9	E1	34	E2	FE	50	D2	:	E2	dde8	55	54	20	21	55	54	44	63	:	3A
DC00	9A	E8	FE	40	D2	7D	E8	21	:	18	ddf0	55	54	49	61	4F	50	20	03	:	15
DC08	15	DC	3D	5F	16	00	19	19	:	D5	ddf8	55	53	48	02	45	53	20	19	:	C3
DC10	7E	23	66	6F	E9	52	E2	45	:	D8	DE00	45	54	20	1E	45	54	49	5B	:	14
DC18	E4	48	E4	88	E4	F5	E4	32	:	87	DE08	45	54	4E	5C	4C	20	20	12	:	E1
DC20	E5	79	E5	3F	E5	7C	E5	82	:	4A	DE10	4C	41	20	4A	4C	43	20	10	:	B6
DC28	E5	7F	E5	85	E5	CC	E5	D9	:	3D	DE18	4C	43	41	49	4C	44	20	59	:	22
DC30	E5	D1	E7	38	E6	3A	E6	3D	:	18	DE20	52	20	20	13	52	41	20	4C	:	A4
DC38	E6	40	E6	43	E6	46	E6	49	:	AA	DE28	52	43	20	11	52	43	41	4B	:	E7
DC40	E6	56	E6	5C	E6	59	E6	B9	:	5C	DE30	52	44	20	5A	53	54	20	1F	:	F6
DC48	E6	17	E7	13	E7	4D	E7	77	:	89	DE38	42	43	20	08	43	46	20	44	:	9A
DC50	E7	BB	E7	FC	E7	32	E8	3A	:	C0	DE40	45	54	20	18	4C	41	20	14	:	92
DC58	9D	EA	3D	28	12	2A	AF	EA	:	C1	DE48	52	41	20	15	52	4C	20	16	:	9C
DC60	11	B2	EA	CD	C2	E0	3E	3A	:	94	DE50	55	42	20	07	4F	52	20	0B	:	8A
DC68	12	21	B1	EA	CD	05	D9	2A	:	A3	DE58	21	5D	DE	18	14	4E	5A	5A	:	8A
DC70	AE	EA	26	00	ED	5B	AC	EA	:	9C	DE60	20	4E	43	43	20	50	4F	50	:	03
DC78	19	DA	56	E9	22	AC	EA	3A	:	24	DE68	45	50	20	4D	20	00	21	92	:	D5
DC80	9E	EA	B7	CA	3C	DB	ED	7B	:	88	DE70	DE	3A	A1	EA	FE	03	30	15	:	E9
DC88	9F	EA	C9	23	7E	B7	C8	FE	:	70	DE78	06	00	11	A2	EA	1A	BE	13	:	8E
DC90	3B	C8	FE	20	C0	23	18	F4	:	10	DE80	23	20	04	1A	BE	28	09	23	:	73
DC98	FE	30	38	0C	FE	3A	38	0A	:	EC	DE88	04	7E	B7	20	ED	3E	FF	C9	:	4C
DCA0	FE	3F	38	04	FE	5B	38	02	:	0C	DE90	78	C9	42	20	43	20	44	20	:	6A
DCA8	37	C9	B7	C9	FE	61	D8	FE	:	B5	DE98	45	20	48	20	4C	20	41	20	:	9A
DCB0	7B	D0	E6	5F	C9	2A	AA	EA	:	17	DEA0	49	20	52	20	41	46	42	43	:	E7
DCB8	7E	FE	2C	C2	5F	E9	CD	8B	:	0A	DEA8	44	45	48	4C	53	50	49	58	:	61
DCC0	DC	22	AA	EA	C9	2A	AA	EA	:	19	DEB0	49	59	00	CD	B5	DC	2A	AA	:	D4
DCC8	CD	8C	DC	C2	5F	E9	C9	3A	:	42	DEB8	EA	7E	FE	28	28	21	CD	54	:	FB
DCD0	A1	EA	FE	02	38	3D	FE	05	:	03	DEC0	DF	2A	AA	EA	7E	FE	29	CA	:	0C
DCD8	30	39	21	A2	EA	7E	D6	41	:	AB	DEC8	59	E9	3A	CE	EA	3C	20	04	:	94
DCE0	FE	1A	30	2F	6F	26	00	11	:	1D	DED0	3E	40	18	6E	3A	CF	EA	B7	:	AE
DCE8	15	DD	19	5E	23	7E	93	28	:	C5	DEDB	C2	65	E9	3A	CE	EA	C9	CD	:	98
DCF0	22	16	00	21	30	DD	19	19	:	98	DEE0	8B	DC	22	AA	EA	CD	54	DF	:	1D
DCF8	19	19	4F	06	03	11	A3	EA	:	28	DEE8	2A	AA	EA	7E	FE	29	C2	59	:	7E
DD00	1A	BE	20	06	13	23	10	F8	:	3C	DEF0	E9	CD	8B	DC	CD	F2	DF	C2	:	7D
DD08	7E	C9	0D	28	06	04	23	10	:	B9	DEF8	65	E9	22	AA	EA	3A	CE	EA	:	F6
DD10	FD	18	E8	AF	C9	00	03	04	:	7C	DF00	FE	FF	20	05	3E	30	C3	42	:	95
DD18	0C	14	19	19	19	1A	21	23	:	C9	DF08	DF	FE	0E	38	04	C6	12	18	:	17
DD20	23	28	28	2A	31	33	33	42	:	76	DF10	31	3A	CF	EA	B7	C2	65	E9	:	EB
DD28	49	49	49	49	49	4A	4A	4A	:	4B	DF18	3A	CE	EA	FE	0C	20	04	3E	:	5E
DD30	44	43	20	06	44	44	20	05	:	5A	DF20	10	18	1F	FE	0A	20	04	3E	:	B1
DD38	4E	44	20	09	49	54	20	17	:	8F	DF28	11	18	17	FE	0B	20	04	3E	:	AB
DD40	41	4C	4C	1D	43	46	20	43	:	E2	DF30	12	18	0F	FE	0D	20	04	3E	:	A6
DD48	50	20	20	0C	50	44	20	56	:	A6	DF38	13	18	07	FE	01	C2	65	E9	:	41
DD50	50	44	52	57	50	49	20	54	:	4A	DF40	3E	14	32	CE	EA	C9	CD	54	:	26
DD58	50	49	52	55	50	4C	20	42	:	3E	DF48	DF	3A	CE	EA	3C	C2	65	E9	:	1D
DD60	41	41	20	41	45	43	20	0E	:	99	DF50	2A	CA	EA	C9	AF	32	CF	EA	:	41
DD68	45	46	42	83	45	46	4D	84	:	AC	DF58	67	6F	22	CA	EA	3D	32	CE	:	E9

DF60	EA	2A	AA	EA	7E	FE	2B	28	:	77	E150	E9	CD	8C	DC	CA	62	E9	11	:	44
DF68	74	FE	2D	28	73	FE	27	20	:	7F	E158	A3	E1	06	04	7E	CD	AC	DC	:	61
DF70	05	CD	7F	E0	18	3C	FE	30	:	B3	E160	EB	BE	EB	C2	5F	E9	13	23	:	D4
DF78	38	09	FE	3A	30	05	CD	FA	:	75	E168	10	F2	CD	8C	DC	CA	5F	E9	:	49
DF80	DF	1B	2F	CD	AC	DC	CD	A0	:	EB	E170	22	AA	EA	2A	A2	EA	E5	2A	:	7B
DF88	DC	38	1B	CD	16	E1	CD	6E	:	2E	E178	A4	EA	E5	2A	A6	EA	E5	CD	:	DF
DF90	DE	3C	28	0D	3D	32	CE	EA	:	76	E180	46	DF	EB	E1	22	A6	EA	E1	:	84
DF98	3A	CF	EA	B7	C2	5C	E9	18	:	C9	E188	22	A4	EA	E1	22	A2	EA	EB	:	2A
DFA0	24	CD	EF	E8	18	0C	FE	24	:	0E	E190	E5	11	BB	EA	CD	FB	E0	3E	:	81
DFA8	C2	65	E9	23	22	AA	EA	2A	:	13	E198	3D	32	C0	EA	E1	CD	C6	E8	:	75
DFB0	AC	EA	EB	2A	CA	EA	3A	CF	:	68	E1A0	C3	C5	DC	45	51	55	20	CD	:	3C
DFB8	EA	B7	FA	C0	DF	19	18	02	:	6D	E1A8	46	DF	CD	C5	DC	2A	CA	EA	:	71
DFC0	ED	52	22	CA	EA	2A	AA	EA	:	D3	E1B0	22	AC	EA	3A	9D	EA	3D	C8	:	7E
DFC8	CD	8C	DC	22	AA	EA	CD	EF	:	A7	E1B8	E5	11	BB	EA	CD	FB	E0	E1	:	24
DFD0	DF	C8	FE	2B	28	07	FE	2D	:	2A	E1C0	AF	57	5F	C3	CB	D8	CD	C5	:	5D
DFD8	28	06	C3	65	E9	3E	01	01	:	7F	E1C8	DC	3E	FF	32	9E	EA	C9	CD	:	69
DFE0	3E	FF	32	CF	EA	CD	8B	DC	:	5C	E1D0	46	DF	CD	13	E9	2A	AA	EA	:	AC
DFE8	22	AA	EA	7E	C3	6D	DF	FE	:	41	E1D8	CD	8C	DC	C8	FE	2C	C2	65	:	4E
DFF0	29	C8	B7	C8	FE	3B	C8	FE	:	6F	E1E0	E9	CD	8B	DC	22	AA	EA	18	:	EB
DFF8	2C	C9	11	A2	EA	06	07	2A	:	C9	E1E8	E6	CD	46	DF	CD	18	E9	2A	:	D0
E000	AA	EA	7E	CD	AC	DC	CD	EF	:	23	E1F0	AA	EA	CD	8C	DC	C8	FE	2C	:	BB
E008	DF	28	14	FE	20	28	10	FE	:	6F	E1F8	C2	65	E9	CD	8B	DC	22	AA	:	10
E010	2B	28	0C	FE	2D	28	08	12	:	CC	E200	EA	18	E6	2A	AA	EA	7E	FE	:	22
E018	13	23	10	E6	C3	68	E9	22	:	62	E208	27	C2	65	E9	23	7E	B7	CA	:	59
E020	AA	EA	EB	2B	7E	FE	44	28	:	92	E210	59	E9	FE	20	DA	65	E9	FE	:	86
E028	05	FE	48	28	29	23	36	00	:	F5	E218	7F	CA	65	E9	FE	27	20	06	:	E2
E030	21	00	00	11	A2	EA	1A	B7	:	8F	E220	23	7E	FE	27	20	07	E5	CD	:	9F
E038	C8	D6	30	FE	0A	30	6A	44	:	B4	E228	2C	E9	E1	18	DF	CD	8C	DC	:	22
E040	4D	29	38	65	29	38	62	09	:	DF	E230	C8	C3	5F	E9	CD	46	DF	CD	:	92
E048	38	5F	29	38	5C	4F	06	00	:	A9	E238	C5	DC	ED	5B	CA	EA	CB	7A	:	E2
E050	09	38	56	13	18	E0	36	00	:	D8	E240	C2	68	E9	2A	AC	EA	19	22	:	0E
E058	21	00	00	11	A2	EA	1A	B7	:	8F	E248	AC	EA	3A	9D	EA	3D	C8	C3	:	1F
E060	C8	D6	30	FE	0A	38	08	D6	:	EC	E250	CB	D8	CD	B6	DE	FE	40	CA	:	0C
E068	11	FE	06	30	3C	C6	0A	4F	:	A0	E258	65	E9	2A	CA	EA	22	CC	EA	:	04
E070	06	00	7C	E6	F0	20	32	29	:	D3	E260	F5	CD	B3	DE	C1	4F	CD	C5	:	F5
E078	29	29	29	09	13	18	DF	ED	:	7B	E268	DC	78	FE	09	38	05	FE	10	:	A6
E080	5B	AA	EA	1A	FE	27	20	21	:	6F	E270	DA	94	E3	FE	07	D2	FB	E2	:	02
E088	06	03	21	00	00	13	1A	B7	:	0E	E278	FE	06	20	01	3C	87	87	87	:	F6
E090	CA	59	E9	FE	20	38	12	FE	:	72	E280	47	79	FE	07	30	0B	FE	06	:	04
E098	7F	28	0E	FE	27	20	06	13	:	13	E288	20	01	3C	C6	40	80	C3	2C	:	D2
E0A0	1A	FE	27	20	07	65	6F	10	:	4A	E290	E9	FE	40	20	09	3E	06	80	:	14
E0A8	E4	C3	68	E9	ED	53	AA	EA	:	CC	E298	CD	2C	E9	C3	13	E9	FE	10	:	AF
E0B0	C9	7D	B7	9F	BC	C4	BA	E0	:	86	E2A0	20	06	3E	46	80	C3	2C	E9	:	02
E0B8	7D	C9	E5	3E	10	CD	2C	EA	:	5C	E2A8	FE	20	28	07	FE	21	20	10	:	9C
E0C0	E1	C9	D5	01	10	27	CD	ED	:	71	E2B0	3E	FD	21	3E	DD	C5	CD	2C	:	35
E0C8	E0	01	E8	03	CD	ED	E0	01	:	67	E2B8	E9	C1	CD	A2	E2	C3	20	E9	:	C7
E0D0	64	00	CD	ED	E0	01	0A	00	:	09	E2C0	78	FE	38	C2	65	E9	79	FE	:	35
E0D8	CD	ED	E0	7D	F6	30	12	13	:	62	E2C8	07	20	05	3E	57	C3	6C	E3	:	D3
E0E0	E1	06	04	7E	FE	30	C0	36	:	8D	E2D0	FE	08	20	05	3E	5F	C3	6C	:	F7
E0E8	20	23	10	F7	C9	3E	30	B7	:	38	E2D8	E3	FE	11	20	05	3E	0A	C3	:	22
E0F0	ED	42	38	03	3C	18	F8	09	:	BF	E2E0	2C	E9	FE	12	20	05	3E	1A	:	A2
E0F8	12	13	C9	7C	CD	00	E1	7D	:	95	E2E8	C3	2C	E9	FE	30	C2	65	E9	:	16
E100	F5	0F	0F	0F	0F	CD	09	E1	:	E8	E2F0	3E	3A	CD	2C	E9	C3	18	E9	:	1E
E108	F1	E6	0F	C6	30	FE	3A	38	:	4C	E2F8	79	FE	07	30	2C	FE	06	20	:	FE
E110	02	C6	07	12	13	C9	21	A2	:	80	E300	01	3C	4F	78	FE	10	20	06	:	38
E118	EA	06	06	3E	20	77	23	10	:	FE	E308	3E	70	81	C3	2C	E9	FE	20	:	25
E120	FC	70	23	70	2A	AA	EA	11	:	CE	E310	28	07	FE	21	20	41	3E	FD	:	EA
E128	A2	EA	7E	CD	AC	DC	CD	98	:	C4	E318	21	3E	DD	C5	CD	2C	E9	C1	:	A4
E130	DC	38	0D	23	4F	04	78	FE	:	0D	E320	CD	08	E3	2A	CC	EA	C3	23	:	7E
E138	07	30	EF	79	12	13	18	EA	:	C6	E328	E9	FE	40	C2	F5	E3	78	FE	:	37
E140	22	AA	EA	78	32	A1	EA	C9	:	B4	E330	10	20	08	3E	36	CD	2C	E9	:	8E
E148	2A	AA	EA	7E	FE	20	C2	62	:	7E	E338	C3	13	E9	FE	20	28	08	FE	:	0B

E340	21	C2	65	E9	3E	FD	21	3E	:	CB	E530	2C	E9	CD	69	E5	20	04	06	:	5A
E348	DD	CD	2C	E9	3E	36	CD	2C	:	2C	E538	88	18	55	06	08	18	0C	CD	:	F4
E350	E9	CD	23	E3	C3	13	E9	79	:	F4	E540	69	E5	20	05	06	98	C3	90	:	64
E358	FE	07	C2	65	E9	78	FE	07	:	92	E548	E5	06	00	FE	0C	C2	65	E9	:	05
E360	20	04	3E	47	18	06	FE	08	:	CD	E550	79	D6	0A	FE	04	D2	65	E9	:	78
E368	20	0C	3E	4F	F5	3E	ED	CD	:	A6	E558	87	87	87	87	80	C6	42	F5	:	99
E370	2C	E9	F1	C3	2C	E9	FE	11	:	ED	E560	3E	ED	CD	2C	E9	F1	C3	2C	:	ED
E378	20	05	3E	02	C3	2C	E9	FE	:	3B	E568	E9	CD	B6	DE	F5	CD	B3	DE	:	9D
E380	12	20	05	3E	12	C3	2C	E9	:	5F	E570	C1	4F	CD	C5	DC	78	FE	06	:	FA
E388	FE	30	C2	65	E9	3E	32	CD	:	78	E578	C9	3E	90	01	3E	A0	01	3E	:	B5
E390	2C	E9	18	71	79	FE	40	20	:	75	E580	A8	01	3E	B0	01	3E	B8	F5	:	83
E398	1F	78	D6	0A	FE	04	30	07	:	B0	E588	CD	B6	DE	C1	4F	CD	C5	DC	:	DF
E3A0	87	87	87	87	3C	18	0B	3E	:	B9	E590	79	FE	07	30	07	FE	06	20	:	D9
E3A8	DD	28	02	3E	FD	CD	2C	E9	:	24	E598	01	3C	18	2C	FE	10	28	26	:	DD
E3B0	3E	21	CD	2C	E9	C3	18	E9	:	05	E5A0	FE	20	28	12	FE	21	28	11	:	B0
E3B8	FE	30	20	18	78	FE	0C	20	:	08	E5A8	FE	40	C2	65	E9	3E	46	80	:	52
E3C0	08	3E	2A	CD	2C	E9	C3	18	:	2D	E5B0	CD	2C	E9	C3	13	E9	3E	DD	:	BC
E3C8	E9	D6	0A	FE	06	D2	65	E9	:	ED	E5B8	21	3E	FD	C5	CD	2C	E9	C1	:	C4
E3D0	16	08	18	49	78	FE	0D	C2	:	C4	E5C0	CD	C6	E5	C3	20	E9	3E	06	:	88
E3D8	65	E9	79	FE	0C	28	11	FE	:	08	E5C8	80	C3	2C	E9	CD	B6	DE	F5	:	AE
E3E0	0E	28	0B	FE	0F	C2	65	E9	:	5B	E5D0	CD	C5	DC	F1	01	03	04	18	:	7F
E3E8	3E	FD	01	3E	DD	CD	2C	E9	:	39	E5D8	0B	CD	B6	DE	F5	CD	C5	DC	:	CF
E3F0	3E	F9	C3	2C	E9	78	FE	30	:	B5	E5E0	F1	01	0B	05	FE	07	30	0C	:	43
E3F8	C2	65	E9	79	FE	0C	20	0E	:	C1	E5E8	FE	06	20	01	3C	87	87	87	:	F6
E400	3E	22	CD	2C	E9	2A	CC	EA	:	22	E5F0	80	C3	2C	E9	FE	10	28	18	:	A6
E408	22	CA	EA	C3	18	E9	D6	0A	:	7A	E5F8	FE	20	28	07	FE	21	20	16	:	A2
E410	FE	06	D2	65	E9	16	00	2A	:	64	E600	3E	FD	21	3E	DD	C5	CD	2C	:	35
E418	CC	EA	22	CA	EA	FE	04	30	:	BE	E608	E9	C1	CD	10	E6	C3	20	E9	:	39
E420	10	87	87	87	87	82	C6	43	:	B7	E610	3E	30	80	C3	2C	E9	D6	0A	:	A6
E428	F5	3E	ED	CD	2C	E9	F1	18	:	0B	E618	FE	06	D2	65	E9	FE	04	30	:	56
E430	0E	3E	DD	28	02	3E	FD	D5	:	63	E620	06	87	87	87	87	18	0D	3E	:	85
E438	CD	2C	E9	D1	3E	22	82	CD	:	62	E628	DD	28	02	3E	FD	C5	CD	2C	:	00
E440	2C	E9	C3	18	E9	3E	04	FE	:	19	E630	E9	C1	3E	20	81	C3	2C	E9	:	61
E448	AF	F5	CD	B6	DE	CD	C5	DC	:	73	E638	AF	01	3E	01	01	3E	02	01	:	31
E450	D1	3A	CE	EA	FE	09	28	1A	:	0C	E640	3E	03	01	3E	04	01	3E	05	:	C8
E458	FE	10	D2	65	E9	D6	0A	DA	:	E8	E648	01	3E	07	F5	CD	B6	DE	C1	:	5D
E460	65	E9	FE	03	CA	65	E9	FE	:	65	E650	4F	CD	C5	DC	18	24	3E	08	:	3F
E468	04	30	0D	87	87	87	87	C6	:	23	E658	01	3E	10	01	3E	18	F5	CD	:	68
E470	C1	01	3E	F1	82	C3	2C	E9	:	4B	E660	B6	DE	C1	FE	40	C2	65	E9	:	A3
E478	3E	DD	28	02	3E	FD	D5	CD	:	22	E668	2A	CA	EA	7D	E6	F8	B4	C2	:	AF
E480	2C	E9	F1	C6	E1	C3	2C	E9	:	85	E670	65	E9	78	85	F5	CD	B3	DE	:	9E
E488	CD	B6	DE	CD	B5	DC	3A	CE	:	C7	E678	C1	4F	79	FE	20	28	07	FE	:	D4
E490	EA	FE	09	20	1A	16	41	CD	:	4F	E680	21	20	16	3E	FD	21	3E	DD	:	CE
E498	EB	E4	16	46	CD	EB	E4	16	:	DD	E688	C5	CD	2C	E9	3E	CB	CD	2C	:	A9
E4A0	27	CD	EB	E4	CD	BC	DC	C2	:	BA	E690	E9	CD	20	E9	C1	0E	06	18	:	AC
E4A8	65	E9	3E	08	C3	2C	E9	F5	:	61	E698	18	0E	06	FE	10	28	08	FE	:	6B
E4B0	CD	B6	DE	CD	C5	DC	C1	3A	:	CA	E6A0	07	D2	65	E9	FE	06	20	01	:	4C
E4B8	CE	EA	4F	78	FE	0B	20	0B	:	B3	E6A8	3C	4F	C5	3E	CB	CD	2C	E9	:	3B
E4C0	79	FE	0C	C2	65	E9	3E	EB	:	BC	E6B0	C1	78	87	87	87	81	C3	2C	:	3E
E4C8	C3	2C	E9	FE	13	C2	65	E9	:	F9	E6B8	E9	2A	AA	EA	7E	FE	28	28	:	73
E4D0	79	FE	0C	28	11	FE	0E	28	:	F0	E6C0	27	E5	CD	95	E7	38	0D	3C	:	D6
E4D8	08	FE	0F	C2	65	E9	3E	FD	:	60	E6C8	CA	65	E9	3D	87	87	87	C6	:	B0
E4E0	01	3E	DD	CD	2C	E9	3E	E3	:	1F	E6D0	C2	E1	18	06	E1	22	AA	EA	:	58
E4E8	C3	2C	E9	7E	CD	AC	DC	BA	:	65	E6D8	3E	C3	F5	CD	46	DF	CD	C5	:	7A
E4F0	C2	65	E9	23	C9	CD	69	E5	:	17	E6E0	DC	F1	CD	2C	E9	C3	18	E9	:	73
E4F8	20	05	06	80	C3	90	E5	FE	:	E1	E6E8	CD	B6	DE	F5	CD	C5	DC	F1	:	B5
E500	0C	28	1E	FE	0E	28	08	FE	:	8C	E6F0	FE	10	28	1A	FE	20	28	08	:	9E
E508	0F	C2	65	E9	3E	FD	21	3E	:	B9	E6F8	FE	21	C2	65	E9	06	FD	21	:	53
E510	DD	C5	CD	2C	E9	C1	79	FE	:	BC	E700	06	DD	2A	CA	EA	7C	B5	C2	:	B4
E518	0C	CA	65	E9	B8	20	02	0E	:	0C	E708	65	E9	78	CD	2C	E9	3E	E9	:	CF
E520	0C	79	D6	0A	FE	04	D2	65	:	9E	E710	C3	2C	E9	3E	10	18	1C	2A	:	84
E528	E9	87	87	87	87	C6	09	C3	:	97	E718	AA	EA	E5	CD	95	E7	38	0D	:	07

E720	FE	04	D2	65	E9	87	87	87	:	B7	E910	00	00	C9	3A	CA	EA	18	14	:	E3
E728	C6	20	E1	18	06	E1	22	AA	:	92	E918	CD	13	E9	3A	CB	EA	18	0C	:	DC
E730	EA	3E	18	F5	CD	46	DF	CD	:	F4	E920	2A	CA	EA	3A	9D	EA	3D	28	:	04
E738	C5	DC	F1	CD	2C	E9	2A	CA	:	68	E928	03	CD	B1	E0	47	3A	9D	EA	:	69
E740	EA	ED	5B	AC	EA	B7	ED	52	:	BE	E930	3D	28	18	3A	AE	EA	FE	04	:	51
E748	2B	2B	C3	23	E9	2A	AA	EA	:	E3	E938	30	0D	87	5F	16	00	21	C0	:	1A
E750	E5	CD	95	E7	38	0D	3C	CA	:	79	E940	EA	19	EB	7B	CD	00	E1	78	:	8C
E758	65	E9	3D	87	87	87	C6	C4	:	AA	E948	CD	AC	D8	21	AE	EA	34	7E	:	BC
E760	E1	18	06	E1	22	AA	EA	3E	:	D4	E950	FE	80	D2	5F	E9	C9	3E	00	:	9F
E768	CD	F5	CD	46	DF	CD	C5	DC	:	22	E958	01	3E	02	01	3E	04	01	3E	:	C3
E770	F1	CD	2C	E9	C3	18	E9	2A	:	C1	E960	06	01	3E	08	01	3E	0C	01	:	99
E778	AA	EA	CD	8C	DC	3E	C9	CA	:	9A	E968	3E	14	CD	2C	EA	C3	57	DC	:	2B
E780	2C	E9	CD	95	E7	D2	5F	E9	:	78	E970	86	E9	97	E9	A5	E9	B6	E9	:	1C
E788	3C	CA	65	E9	3D	87	87	87	:	26	E978	C3	E9	CF	E9	E6	E9	F4	E9	:	10
E790	C6	C0	C3	2C	E9	CD	16	E1	:	22	E980	00	EA	10	EA	20	EA	41	64	:	93
E798	CD	58	DE	47	2A	AA	EA	CD	:	D5	E988	64	72	65	73	73	20	4F	76	:	06
E7A0	8C	DC	22	AA	EA	28	11	FE	:	55	E990	65	72	66	6C	6F	77	00	42	:	D1
E7A8	2C	C2	5F	E9	CD	88	DC	22	:	8C	E998	61	6C	61	6E	63	65	20	45	:	C9
E7B0	AA	EA	CA	65	E9	78	B7	C9	:	A4	E9A0	72	72	6F	72	00	45	78	70	:	F2
E7B8	78	37	C9	CD	46	DF	CD	C5	:	FC	E9A8	72	65	73	73	69	6F	6E	20	:	23
E7C0	DC	2A	CA	EA	7D	E6	C7	B4	:	9B	E9B0	45	72	72	6F	72	00	46	6F	:	BF
E7C8	C2	65	E9	3E	C7	B5	C3	2C	:	B9	E9B8	72	6D	61	74	20	45	72	72	:	FD
E7D0	E9	CD	46	DF	CD	C5	DC	2A	:	73	E9C0	6F	72	00	4C	61	62	65	6C	:	C1
E7D8	CA	EA	7C	B7	C2	65	E9	7D	:	74	E9C8	20	45	72	72	6F	72	00	4D	:	77
E7E0	FE	03	D2	65	E9	F5	3E	ED	:	41	E9D0	75	6C	74	69	70	6C	79	20	:	33
E7E8	CD	2C	E9	F1	06	46	B7	28	:	FE	E9D8	44	65	66	69	6E	65	64	20	:	CF
E7F0	07	06	56	3D	28	02	06	5E	:	2E	E9E0	4C	61	62	65	6C	00	4F	70	:	9F
E7F8	78	C3	2C	E9	CD	69	E5	20	:	8B	E9E8	65	72	61	6E	64	20	45	72	:	E1
E800	17	04	79	FE	30	20	17	3E	:	37	E9F0	72	6F	72	00	50	68	61	73	:	DF
E808	DB	CD	2C	E9	2A	CA	EA	7C	:	17	E9F8	65	20	45	72	72	6F	72	00	:	8F
E810	B7	C2	65	E9	7D	C3	2C	E9	:	1C	EA00	52	65	66	65	72	65	6E	63	:	2A
E818	FE	06	D2	65	E9	79	FE	14	:	AF	EA08	65	20	45	72	72	6F	72	00	:	8F
E820	C2	65	E9	C5	3E	ED	CD	2C	:	F9	EA10	55	6E	64	65	66	69	6E	65	:	2E
E828	E9	F1	87	87	87	C6	40	C3	:	38	EA18	64	20	4C	61	62	65	6C	00	:	64
E830	2C	E9	CD	B6	DE	2A	CA	EA	:	54	EA20	56	61	6C	75	65	20	45	72	:	D4
E838	E5	F5	CD	B3	DE	C1	4F	CD	:	15	EA28	72	6F	72	00	6F	26	00	11	:	F9
E840	C5	DC	E1	22	CA	EA	79	FE	:	CF	EA30	70	E9	19	5E	23	56	1A	D5	:	38
E848	06	20	17	0C	78	FE	30	20	:	0F	EA38	32	B9	EA	21	67	EA	3E	FF	:	84
E850	17	3E	D3	CD	2C	E9	2A	CA	:	FE	EA40	CD	2E	D9	2A	AF	EA	11	D0	:	78
E858	EA	7C	B7	C2	65	E9	7D	C3	:	6D	EA48	EA	CD	C2	E0	AF	12	21	D0	:	0B
E860	2C	E9	FE	06	D2	65	E9	78	:	B1	EA50	EA	CD	2E	D9	21	6B	EA	AF	:	E3
E868	FE	14	C2	65	E9	C5	3E	ED	:	12	EA58	CD	2E	D9	2A	D6	EA	23	22	:	03
E870	CD	2C	E9	C1	79	87	87	87	:	B1	EA60	D6	EA	E1	AF	C3	2E	D9	0D	:	27
E878	C6	41	C3	2C	E9	D6	40	5F	:	54	EA68	0A	20	00	3A	20	20	00	00	:	A4
E880	16	00	21	8D	E8	19	7E	CD	:	10	EA70	00	00	00	00	00	00	00	00	:	00
E888	2C	E9	C3	C5	DC	D9	27	2F	:	A8	EA78	00	00	00	00	00	00	00	00	:	00
E890	3F	37	00	76	F3	FB	07	17	:	FB	EA80	00	00	00	00	00	00	00	00	:	00
E898	0F	1F	D6	50	5F	16	00	21	:	EA	EA88	00	00	00	00	00	00	00	00	:	00
E8A0	B1	E8	19	7E	F5	3E	ED	CD	:	1D	EA90	00	00	00	00	00	00	00	00	:	00
E8A8	2C	E9	F1	CD	2C	E9	C3	C5	:	70	EA98	00	00	00	00	00	00	00	00	:	00
E8B0	DC	A0	B0	A8	B8	A1	B1	A9	:	87	EAA0	00	00	00	00	00	00	00	00	:	00
E8B8	B9	44	6F	67	4D	45	A2	B2	:	B9	EAA8	00	00	00	00	00	00	00	00	:	00
E8C0	AA	BA	A3	B3	AB	BB	22	AB	:	EA	EAB0	00	00	00	00	00	00	00	00	:	00
E8C8	EA	3A	9D	EA	3D	20	0C	21	:	35	EAB8	00	00	00	00	00	00	00	00	:	00
E8D0	A2	EA	CD	A6	D9	B7	C8	3E	:	95	EAC0	00	00	00	00	00	00	00	00	:	00
E8D8	0A	18	11	E5	21	A2	EA	CD	:	92	EAC8	00	00	00	00	00	00	00	00	:	00
E8E0	92	D9	2A	AB	EA	D1	B7	ED	:	9C	EAD0	00	00	00	00	00	00	00	00	:	00
E8E8	52	C8	3E	0E	C3	2C	EA	21	:	60	EAD8	00	00	00	00	00	00	00	00	:	00
E8F0	A2	EA	3A	9D	EA	3D	20	0A	:	B4	EAE0	00	00	00	00	00	00	00	00	:	00
E8F8	CD	92	D9	2A	AB	EA	B7	C8	:	73	EAE8	00	00	00	00	00	00	00	00	:	00
E900	18	0D	CD	92	D9	2A	AB	EA	:	19	EAf0	00	00	00	00	00	00	00	00	:	00
E908	B7	C8	3E	12	CD	2C	EA	21	:	D3	EAf8	00	00	00	00	00	00	00	00	:	00

索引

A	ADD命令	70
B	Bコマンド	31
	BASICインタプリタ	25
	BREAKX(BIOS)	131
C	CALL命令	86, 96
	CHGCAP(BIOS)	134
	CHGET(BIOS)	128
	CHPUT(BIOS)	122
	CP命令	100
	CPU	120
D	Dコマンド	32
	DEC命令	81
	DEFS命令	127
	DEFB命令	127
	DEFM命令	127
	DEFW命令	127
E	EQU命令	65
	END命令	52
G	Gコマンド	35
	GTSTCK(BIOS)	132
	GTTRIG(BIOS)	132
I	I/O	120
	I/Oポート	120
	INC命令	81
J	JP命令	85, 95
	JR命令	91, 95
L	LD命令	50
O	ORG命令	52
P	POP命令	61
	POSIT(BIOS)	129
	PUSH命令	61
R	Rコマンド	36
	RAM	14
	RDVRM(BIOS)	136
	RET命令	86, 96
	ROM	14
S	Sコマンド	33
	SUB命令	73
V	VRAM	14
W	WRTVRM(BIOS)	136
X	Xコマンド	34

あ	アスキーコード.....	124
	アセンブラ.....	22, 28
	アセンブリ言語.....	22
	アセンブル.....	22
	アセンブルエラー.....	43
	アドレス.....	12
	オブジェクトプログラム.....	22
	オペランド.....	49
か	カウンタ.....	110
	キャラクタコード.....	123
さ	主命令.....	49
	16進数.....	7
	スタック.....	61, 86
	スタックエリア.....	63
	絶対ジャンプ.....	91
	ソースプログラム.....	22
	相対ジャンプ.....	91
	ソフトウェア.....	120
た	退避.....	61
	ダンプリスト.....	13
	チェックサム.....	46
な	ニーモニック.....	22
	2進数.....	7
	2の補数表現.....	92
	入出力装置.....	118
は	ハードウェア.....	120
	バイト.....	9, 16
	ハンドアセンブル.....	22
	ビット.....	9
	復帰.....	61
	フラグ.....	75
	暴走.....	115
ま	マシン語モニタ.....	28
	メインRAM.....	15
	メモリ.....	12, 120
	メモリマップ.....	15
	モニタアセンブラ.....	28, 38
ら	ラベル.....	65, 88
	レジスタ.....	17
	レジスタ間接.....	59
	レジスタペア.....	55

くじけちゃいけない
マシン語入門

1986年9月8日 初版発行
定価680円

著 者 平塚憲晴
発行者 塚本慶一郎
発行所 株式会社 **アスキー**

〒107 東京都港区南青山6-11-1 スリーエフ南青山ビル
振 替 東京4-161144
TEL (03)486-7111 (大代表)
出版営業部TEL (03)486-1977 (ダイヤルイン)

本書は著作権法上の保護を受けています。本書の一部あるいは全部について(ソフトウェア及びプログラムを含む)、株式会社アスキーから文書による許諾を得ずに、いかなる方法においても無断で複写、複製することは禁じられています。

編集担当 秋山耕一
CTS 株式会社電算プロセス
印刷 株式会社加藤文明社印刷所

ISBN4-87148-028-3 C3055 ¥680E

MSXとAVの素敵な関係

MSX スーパーAV活用法

—パソコンで創る映像と音楽—

●定価1,200円
●AB版●208ページ

ゲームに勝る

究極の活用法を今ここに公開!

MSXは単なるゲームマシンではない。規格が定められた唯一のパソコンとして、今もっとも発展性のあるマシンといえるだろう。その一端を示すのが本書のテーマでもあるAV。MSXとビデオ、シンセサイザなどのAV機器を組み合わせることによって、これまで高価な機材を持つプロにしかできなかったクリエイティブな編集作業がいとも簡単にできてしまうのだ。本書は、A・V・Cのノウハウを初めて紹介したハイテクマニア必読の書である。

〈Visual World〉

- MSXだけでできるグラフィックス
- VTR、ビデオカメラとの組合せ
- 編集VTRとの組合せ
- TVフォト、一眼レフカメラとの組合せ
- ビデオディスクとの組合せ
- プログラムで作る本格的C.G.

〈Music World〉

- MSXだけでできるミュージック
- FM音源を組み込む
- MSXによる自動演奏/編曲システム
- シンセサイザの接続
- リズムマシンの追加
- 究極のコンピュータ・ミュージック・システム

その他、HOW to VIDEO ART,
保存版MSX製品リストなど

情報満載!

■問い合わせ先

〒107東京都港区南青山6-11-1 スリーエフ南青山ビル (株)アスキー 出版営業部 TEL 03(486)1977

1986 Autumn

ASCII

860820

注文方法

〈直接注文の場合〉

・小社に直接ご注文の際は、定価と送料(ポケットバンクシリーズは200円)を加えた金額を現金書留か郵便振替(東京4-161144)で、注文書名、注文冊数、住所、氏名、電話番号を明記の上、お送りください。お申し込みから商品がお手元に届くまでおよそ1～2週間かかります。なお、総額200,000円以上のご注文の場合、現金書留では送金ができませんのでご注意ください。

〈書店注文の場合〉

・書店にご注文の際は、下の注文書に書名と定価をご記入の上お近くの書店にお渡しください。一枚の注文書に何タイトルご記入くださっても結構です。

○注文書○

の部分のみご記入ください。

ご記入の際は、上記の注文方法(書店注文の場合)をよくお読みください。

書店印	注文数	発行所 株式会社 アスキー		
	冊	書名	定価	円
		注文日 月 日 注文主 様		

書店様へ
お手数ですが複数タイトルご注文の際は、各タイトルごとに別の注文書にご記入くださいますようお願いいたします。

・総合図書目録をご希望の方は、小社出版営業部までご連絡ください。

このカタログに記載されている価格、発行予定などは1986年8月20日現在のものです。万一、記載事項に変更がありました節はご了承ください。

株式会社 **アスキー**

〒107 東京都港区南青山6-11-1 スリーエフ南青山ビル

出版営業部

TEL: 03-486-1977

MSXポケットバンクシリーズ〈新刊〉

これだけでわかつちゃう
●**新・MSXの基礎知識**
浅井敬太郎著 定価580円

いままで、なかったことが不思議なくらい。MSXのすべてがわかるポケットバンク遂に登場。普段から疑問に思いながらも、雰囲気ではわかったつもりでいることばや、ホントにむずかしくってわからない専門用語・基礎知識など、MSXのキーワードを理解できます。

すがやみつるの
●**すぐできるパソコン通信**
すがやみつる・オレンジ企画著 定価580円

ゲーム・フリークスのキミもかなり気になる話題沸騰ノのパソコン通信。アクセスしたくてウズウズしてた？ でも、どうやったらつながるのかわからないって？ ご安心あれ。この一冊がどんな質問にも答えてしまう。キミのMSXがますますおもしろくなってきた。

くじけちゃいけない
●**マシン語入門**
平塚憲晴著 定価680円

これから、ひとつマシン語でもモノにするか、などと大志を抱いている方に朗報です。いままでたくわえてきた(ちょっとカジった?) BASICの基礎があれば、身につくマシン語が学べます。そして、やる気はあるけれどギブアップしてしまった人も、もう大丈夫です。

BASICからマシン語を打ち込む
●**おもしろゲームブック**
BASICからマシン語を打ち込む
BITS 著 定価580円

ちょっと長いプログラムだと、めげてしまう人。MSXを買ったばかりで、とりあえず、なんかおもしろいプログラムを打ち込んでみたいな、なんて思っている人。そんなあなたに贈ります。すぐにたのしめるゲームが約20本も入って580円ノのショートプログラム集。

MSX2テクニカルハンドブック

アスキーマイクロソフトFE監修 定価3500円(送料300円)

MSXグラフィック・ワークブック

桜田幸嗣・蓑島 聡共著 定価1500円(送料300円)

MSXスーパーAV活用法

アスキー書籍編集部編 定価1200円(送料300円)

MSXマシン語入門講座

湯浅 敬著 定価1600円(送料300円)

MSXビギナーズ BASIC

児玉真之著 定価1500円(送料300円)

MSX2大研究

MSXマガジン編集部編著 定価680円(送料250円)

MSXポケットバンクシリーズ〈既刊〉

●**アニメC. G.に挑戦ノ**
川野名 勇ノ牧山慶士共著 定価480円

●**マイコン・ジュークボックス**
森田信也ノ伊君高志共著 定価480円

●**BASICゲーム教室**
安田吾郎著 定価480円

●**マイコン・サウンドパック**
工藤賢司著 定価480円

●**ゲームキャラクタ操縦法**
横溝和宏著 定価480円

●**トランプゲーム集**
ポケットバンク編集部編著 定価480円

●**面白パズルブック**
藤沢幸隆ノ桜田幸嗣共著 定価480円

●**プログラムD. J.**
アスキー南国放送局編著 定価480円

●**グラフィックス秘伝**
安田吾郎著 定価480円

●**マイコン野球中継'84**
永谷 脩著 定価480円

●**とにかく速いマシン語ゲーム集**
ポケットバンク編集部編著 定価480円

●**アクションゲーム38**
ぐるーぷ・アレフ著 定価480円

●**知能ゲーム38**
ぐるーぷ・アレフ著 定価480円

●**必殺・ビデオ活用法**
ポケットバンク編集部編著 定価480円

●**占っちゃうからノ**
ポケットバンク編集部編著 定価480円

●**エラー撃退ミニ事典**
ポケットバンク編集部編著 定価480円

